Software Testing

软件测试丛书

软件测试面试突击

——为自己赢得一份测试工程师职位

Software Testing Interview Guide

赵思奇 蔡芸 于春智 编著

- 5大主流测试应用技术，测试计划的制订、软件测试用例、测试工具应用、软件测试开发、英文测试文档编写
- 涵盖软件测试面试17个方面，如数据库，C、C++、Java、C#和Python语言，主流的测试工具，项目测试，Linux和Windows，质量管理等
- 250道经典面试题，全面解析IT公司的面试之道

人民邮电出版社
POSTS & TELECOM PRESS

Software Testing Interview Guide

软件测试面试突击

——为自己赢得一份测试工程师职位



赵思奇

毕业于英国格拉斯哥大学电子与电气工程系。先后就职于IBM、摩托罗拉、汤森路透等公司，软件测试高级工程师。从事过嵌入式软件、金融网站、C/S系统测试等，在黑盒、白盒、灰盒，安全，性能，自动化测试等方面具有丰富的实践经验。在摩托罗拉任职期间多次获得Bug查找月冠军。在专业刊物上发表论文多篇。



蔡芸

毕业于英国阿伯泰邓迪大学软件工程专业。先后就职于微软、摩托罗拉、汤森路透等公司，现就职于Adobe系统软件公司。担任某著名互联网动画、视频播放器制作工具软件的中国测试团队负责人。在工作中奉行最大程度实现测试理论与实际工作相结合，测试能力和开发能力并举的工作方式。从事过大型网站、著名通讯软件、手机软件、金融软件、互联网动画及视频播放器制作工具软件的测试。



于春智

毕业于北京大学计算机软件与微电子学院嵌入式专业，现就职于IGT科技开发（北京）有限公司，从事支付系统软件开发。曾就职于摩托罗拉，先后从事过手机嵌入式开发、软件集成和软件自动化测试工具的开发等工作。

装帧设计：董志桢

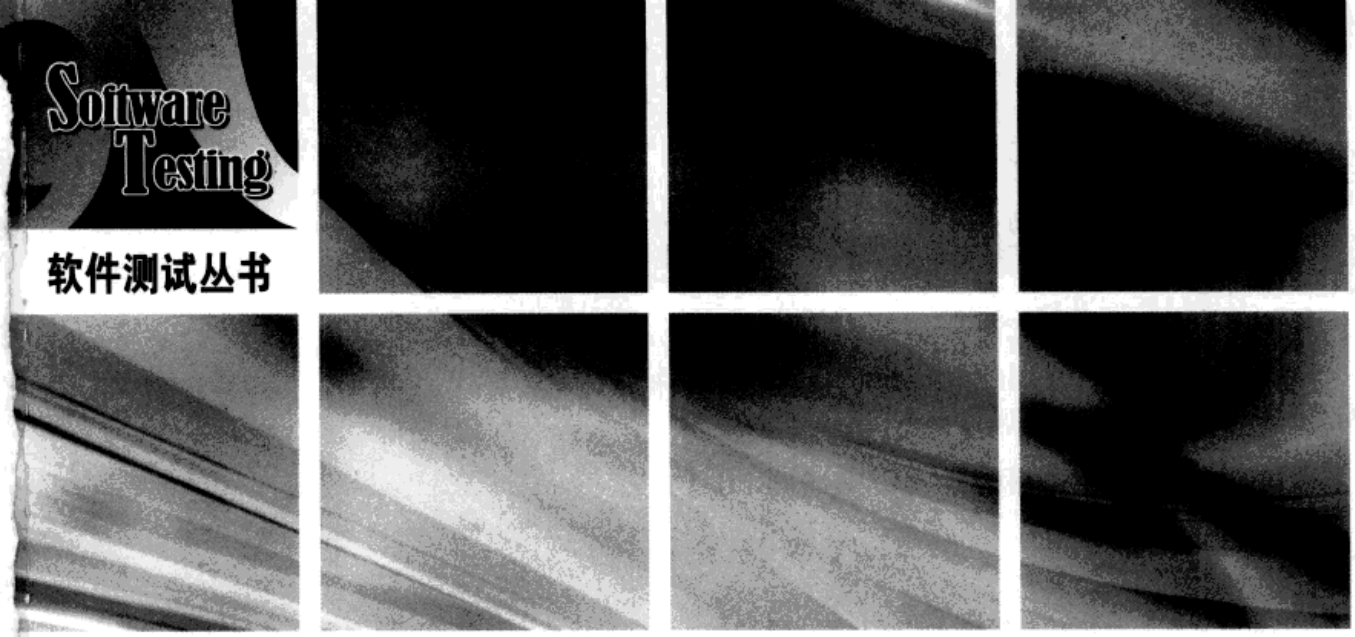
分类建议：计算机/软件工程

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-23092-8

ISBN 978-7-115-23092-8

定价：49.00 元



软件测试丛书

软件测试面试突击

——为自己赢得一份测试工程师职位

Software Testing Interview Guide

赵思奇 蔡芸 于春智 编著

人 民 邮 电 出 版 社

北 京

前 言

本书是一本集面试题和笔试题精解、测试开发指导、项目测试实战技术和技巧、英文面试实战以及帮助软件测试工程师求职的书籍。本书的亮点在于，不但讲解了面试题选，还大篇幅讲解了测试开发、白盒测试和软件项目测试中需要重点掌握的知识和技术。书中给出的题目囊括了软件测试相关的 17 个方面，如质量管理、数据库、Java 语言、主流自动测试工具、项目测试、C#语言、Linux 等，同时还具体介绍了测试实战技术和技巧，如集成工具的开发、测试计划的制订、自动化测试用例与工具开发。另外，书中还附有顶级 IT 企业的 12 个中文、英文软件测试文档模板。

全书分为测试生存篇、测试理论篇、测试开发篇、测试实战篇和测试英文篇，共 5 篇 9 章。本书的组织方式独具匠心，一切为了实战，所有章节都是围绕提出问题→解答问题→分析问题来组织的，切实帮助读者查找不足，提升实战能力及应试技巧。全书共涉及约 250 道各大 IT 企业经典面试题，大部分都是笔者与好友们亲历的各大 IT 公司的真题。

全书主要内容如下。

第一部分测试生存篇，包括第 1 章～第 2 章，主要讲解了软件测试面试概述、软件测试面试前的准备等内容。

第二部分测试基础篇，包括第 3 章～第 4 章，主要讲解了软件测试的基础，如软件测试的背景与概述、测试活动的开展、黑盒测试及测试用例设计方法、测试用例场景模型、C++语言知识考查、C 语言知识考查、Java 语言知识考查、UNIX Shell 知识考查、Linux 知识考查、数据库知识考查、主流测试工具知识考查等。

第三部分测试开发篇，包括第 5 章～第 6 章，主要讲解了 Perl 语言考查、Perl 单元测试、软件测试面试 Python 语言考查等。

第四部分测试实战篇，包括第 7 章～第 8 章，主要讲解了嵌入式软件测试之手机测试实战、网络测试实战——数据传输、网站测试实战等，包括绩效的制订、项目的完整计划制订、集成工具的开发过程、网络测试工具的开发过程、灰盒测试过程（Crontab）、基于手机自动化工具的测试用例开发等。

第五部分测试英文篇，包括第 8 章～第 9 章，主要讲解了典型英文面试笔试题、软件测试面试英文题必知必会基础、软件测试常用文档模板、面向过程的白盒测试用例中文和英文模板、性能用例中文和英文模板、单元测试报告模板等。首先讲述了提高英文的方法（分为听、说、读、写、译 5 方面），然后用英文给出了 4 节面试题（基本理论、相关计算机领域、项目经验、其他），用英文回答，但同时提问和回答都有中文翻译，而且每道题都有详细的分析。本篇最

Offer!

后是12个常用文档模板，母版来自顶级的IT外企（经修订）。

本书由赵思奇、蔡芸、于春智主编，参与编写的还有刘震（汤森路透测试经理）、贾小龙（汤森路透主任级测试工程师）、陈磊（阿里巴巴测试工程师）、周雷（爱立信开发工程师）、周二胜（中国电信工程师）、罗文龙（重庆邮电大学）、陈曼（IGT测试工程师）、黄亚玲、刘琪、黄斌。

本书适合计算机及相关专业学生、软件测试从业者、软件开发人员、软件公司领导及对软件测试有兴趣的读者阅读，联系邮箱为zhangtao@ptpress.com.cn。

编者

Offer!

图书在版编目（CIP）数据

软件测试面试突击 ：为自己赢得一份测试工程师职位 / 赵思奇，蔡芸，于春智编著. -- 北京 ：人民邮电出版社，2010.7
ISBN 978-7-115-23092-8

Ⅰ. ①软… Ⅱ. ①赵… ②蔡… ③于… Ⅲ. ①软件－测试－基本知识 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2010)第091387号

内 容 提 要

本书是一本集面试笔试题精解、测试开发指导、项目测试实战技术和技巧、英文面试实战以及帮助软件测试工程师求职的书籍。本书的亮点在于大篇幅讲解测试开发、白盒测试等。书中给出的面试题囊括了软件测试相关的 17 个方面，如质量管理、数据库、C、C++和 Java 语言、主流自动测试工具、项目测试、Windows、UNIX 和 Linux 等，同时还具体介绍了测试实战技术和技巧，如集成工具的开发、测试计划的制订、自动化测试用例与工具开发。另外，书中还附有顶级 IT 外企的 12 个中文英文软件测试文档模板（经修订）。

全书共分 5 部分，分别为测试生存篇、测试基础篇、测试开发篇、测试实战篇、测试英文篇。本书的组织方式独具匠心，一切为了实战。全书共涉及约 250 道经典面试题，大部分都是笔者与好友们亲历的各大 IT 公司的真题。

本书适合计算机及相关专业学生、软件测试从业者、开发人员、软件公司领导及对软件测试有兴趣的读者阅读。

软件测试面试突击——为自己赢得一份测试工程师职位

◆ 编　　著　赵思奇　蔡　芸　于春智
责任编辑　张　涛

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：24.25
字数：545 千字　　2010 年 7 月第 1 版
印数：1－3 000 册　　2010 年 7 月北京第 1 次印刷

ISBN 978-7-115-23092-8

定价：49.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

目 录

第一部分 测试生存篇

第 1 章 软件测试面试概述……2

1.1 软件测试有前途吗……2
1.2 测试比开发水平差吗……3
1.2.1 初级测试工程师……3
1.2.2 中级测试工程师……4
1.2.3 高级测试工程师……4
1.3 软件测试工程师能挣多少钱……4
1.3.1 黑盒测试……5
1.3.2 测试开发……5
1.3.3 性能测试……5
1.3.4 白盒测试……5
1.3.5 安全测试……5
1.4 职业培训有用吗……6
1.4.1 高校教育与职业培训……7
1.4.2 为非专业人士和学生打开一道门……7
1.4.3 有工作经验者获益最大……7
1.5 大专生能进 IT 外企吗……8
1.5.1 市场概况……8
1.5.2 参加职业培训增强竞争力……8
1.5.3 进入 IT 外企所需的基本条件……8
1.5.4 如何提升自己……9
1.6 不擅长不喜欢开发就适合做测试……9
1.6.1 判定缺陷间的重复及依赖关系需要开发能力……9
1.6.2 使用自动化测试工具需要开发能力……10
1.6.3 黑盒测试偏爱开发能力……10
1.6.4 白盒测试需要开发能力……11
1.6.5 安全测试需要开发能力……11
1.6.6 开发测试工具……12
1.7 有效沟通……12
1.7.1 沟通方式之思辨……12
1.7.2 When and How……13

第 2 章 软件测试面试前的准备……15

2.1 简历要点……15
2.1.1 注意事项……15
2.1.2 简历格式模板……18
2.2 面试要点……20
2.2.1 电话面试……20
2.2.2 笔试……20
2.2.3 面试……20
2.2.4 其他注意事项……21
2.3 考查范围和题型……22
2.3.1 考查范围……22
2.3.2 题型……22

第二部分 测试基础篇

第 3 章 软件测试基础 1——概念与原理 27

3.1 软件测试的背景与概述 27
3.1.1 测试需要什么样素质的测试工程师 27
3.1.2 软件质量与软件缺陷 28
3.1.3 我们创造了什么 29
3.1.4 开发模式中测试阶段 30
3.2 测试活动的开展 31
3.2.1 测试计划 31
3.2.2 测试用例的产生 33
3.2.3 开始执行测试 37
3.2.4 报告测试缺陷 38
3.3 黑盒测试及测试用例设计方法 42
3.3.1 黑盒测试定义 42
3.3.2 黑盒测试方法 47
3.4 白盒测试及测试用例设计方法 69
3.4.1 白盒测试定义 69
3.4.2 白盒测试方法 71
3.5 单元测试 84
3.6 集成测试 88
3.7 系统测试 92
3.8 用户验收测试 96
3.9 回归测试 100
3.10 冒烟测试 103
3.11 α、β、γ 测试 105
3.12 性能测试 1——性能指标 107
3.13 性能测试 2——负载测试 110
3.14 安全性测试 112
3.15 兼容性测试 118
3.16 易用性测试 120
3.17 测试用例场景模型 12 例 123
3.17.1 边界值测试 124
3.17.2 安装和卸载测试 124
3.17.3 表单测试 125
3.17.4 链接测试 125
3.17.5 界面集成测试 126
3.17.6 窗口界面测试 126
3.17.7 菜单界面测试 126
3.17.8 单选框/复选框测试 126
3.17.9 超长字串 Bug 127
3.17.10 新建-编辑接口测试 127
3.17.11 用户登录的 SQL 注入测试 128
3.17.12 SQL 查询语句测试 128
3.17.13 面试题 129

第 4 章 软件测试基础 2——计算机应用 131

4.1 概述 131
4.1.1 难为无米之炊 132
4.1.2 计算机应用领域汇总 132
4.2 C++语言知识考查 133
4.2.1 常见考点汇总 134
4.2.2 经典考题及解析 134
4.3 C 语言知识考查 137
4.3.1 常见考点汇总 138
4.3.2 经典考题及解析 139
4.4 Java 语言知识考查 146
4.4.1 常见考点汇总 147
4.4.2 经典考题及解析 148
4.5 UNIX Shell 知识考查 155

Offer

4.5.1 常见考点汇总 155
4.5.2 经典考题及解析 156
4.6 网络知识考查 159
4.6.1 常见考点汇总 160
4.6.2 经典考题及解析 160
4.7 质量管理知识考查 164
4.7.1 常见考点汇总 166
4.7.2 经典考题及解析 166
4.8 Linux 知识考查 171
4.8.1 常见考点汇总 171
4.8.2 经典考题及解析 172
4.9 VB/VBScript/JavaScript 语言知识考查 175
4.9.1 常见考点汇总 176
4.9.2 经典考题及解析 176
4.10 HTML/XML 知识考查 180
4.10.1 常见考点汇总 180
4.10.2 经典考题及解析 181
4.11 Win32 开发知识考查 183
4.11.1 常见考点汇总 184
4.11.2 经典考题及解析 184
4.12 C#语言知识考查 187
4.12.1 常见考点汇总 188
4.12.2 经典考题及解析 189
4.13 数据库知识考查 197
4.13.1 常见考点汇总 197
4.13.2 经典考题及解析 198
4.14 主流测试工具知识考查 201
4.14.1 常见考点汇总 202
4.14.2 经典考题及解析 203

第三部分 测试开发篇

第 5 章 软件测试面试 Perl 语言考查 ·····211

5.1 Perl 语言考查 211
5.1.1 Perl 文件处理 211
5.1.2 正则表达式 213
5.2 Perl 测试起步 215
5.2.1 安装及运行 215
5.2.2 构建测试用例 218
5.3 Perl 测试管理 219
5.3.1 组织测试 219
5.3.2 测试代码覆盖率 220
5.3.3 编写并测试“测试库” 221
5.4 Perl 单元测试 224
5.4.1 编写测试用例 225
5.4.2 测试方法的类型 228
5.4.3 SKIPPED 和 TODO 测试 229
5.5 Perl 数据库测试 231
5.5.1 装载并测试数据库 231
5.5.2 使用临时数据库 234
5.5.3 模拟数据库 238
5.6 Perl 数据驱动测试框架 240
5.6.1 Test::Base 框架 241
5.6.2 Test::Base 主要函数 243
5.6.3 Test::Base 测试规范 244
5.6.4 Test::Base 过滤器 245

第 6 章 软件测试面试 Python 语言考查 247

6.1 Python 语言考查 247
6.1.1 Python 总体考查 247
6.1.2 Python 测试相关 249
6.2 Python 单元测试 251
6.2.1 Python 单元测试环境搭建 251
6.2.2 PyUnit 测试框架的应用 251

Offer

6.3 py.test 测试框架.....254
6.3.1 py.test 的特点.....254
6.3.2 py.test 框架插件.....255
6.3.3 Funcargs 机制.....256
6.4 Python 综合测试.....257
6.4.1 Python 数据库测试.....257
6.4.2 Python 网页测试.....258
6.4.3 Python 代码覆盖率测试.....261

第四部分 测试实战篇

第 7 章 软件测试面试项目实战.....264

7.1 面试中的项目经验考查.....264
7.1.1 常见考点汇总.....265
7.1.2 经典考题例析.....265
7.2 嵌入式软件测试之手机测试实战 1.....266
7.2.1 测试计划.....267
7.2.2 绩效评定.....271
7.2.3 自动化测试工具 FlexAuto.....274
7.3 嵌入式软件之手机测试实战 2.....279
7.3.1 项目需求概述.....279
7.3.2 PyATA 自动化测试工具.....280
7.4 网络测试实战——数据传输.....283
7.4.1 金融软件特性和测试重点.....283
7.4.2 测试活动的开展.....285
7.4.3 网络数据传输类的软件测试阶段.....286
7.4.4 网络数据传输类的软件测试自动化.....286
7.5 网站测试实战.....289
7.5.1 Jmeter 性能测试.....289
7.5.2 测试流程与计划.....294
7.5.3 测试策略举例.....296

第五部分 测试英文篇

第 8 章 典型英文面试笔试题.....303

8.1 如何提高英文水平.....303
8.1.1 通过听收音机和做听力题来练习听力.....303
8.1.2 通过背诵、聊天、写作来练习口语.....304
8.1.3 通过背单词、看报纸来练习阅读.....305
8.1.4 通过背诵和请高手修改来练习写作.....306
8.1.5 通过范例与做题来练习翻译.....306
8.2 软件测试面试英文题必知必会基础.....307
8.3 软件测试面试英文基础——计算机应用.....312
8.4 项目经验.....320
8.5 软件测试面试英文、能力、心理测试.....325
8.5.1 翻译.....325
8.5.2 性格心理考查.....327
8.5.3 智力考查.....328
8.5.4 能力考查.....329

Offer

第 9 章 软件测试常用文档模板……330

9.1 测试计划模板……330
9.1.1 中文模板……331
9.1.2 表格项解释……334
9.1.3 英文模板……335
9.2 性能测试计划模板……338
9.2.1 中文模板及表格项解释……339
9.2.2 英文模板……345
9.3 黑盒测试用例中、英文模板……351
9.3.1 中文模板……352
9.3.2 表格项解释……352
9.3.3 英文模板……353
9.4 面向过程的白盒测试用例中、英文模板……353
9.4.1 中文模板……354
9.4.2 表格项解释……355
9.4.3 英文模板……355
9.5 面向对象的白盒类测试计划与用例中、英文模板……356
9.5.1 中文模板……357
9.5.2 表格项解释……357
9.5.3 英文模板……358
9.6 性能用例模板……358
9.6.1 中文模板及表格项解释……359
9.6.2 英文模板……362
9.7 缺陷记录报告模板……365
9.7.1 中文模板……366
9.7.2 表格项解释……366
9.7.3 英文模板……367
9.8 例会记录模板……368
9.8.1 中文模板……368
9.8.2 英文模板……369
9.9 工作日志模板……369
9.9.1 中文模板……369
9.9.2 表格项解释……370
9.9.3 英文模板……370
9.10 周报模板……370
9.10.1 中文模板……371
9.10.2 表格项解释……371
9.10.3 英文模板……371
9.11 单元测试报告模板……372
9.11.1 中文模板及各项解释……372
9.11.2 英文模板……374
9.12 测试总结模板……374
9.12.1 中文模板及表格项解释……376
9.12.2 英文模板……377



第一部分

测试生存篇

大家可能平时从媒体上看到过测试职业培训机构的广告，或者可能从从业的亲友那里听说过一些业内的情况，已经对软件测试有一些初步的了解。或者有的读者刚刚涉足这个行业，对软件测试有一些模模糊糊的认识。

软件测试有前途吗？测试比开发水平差？软件测试能挣多少钱？留学毕业怎么进入 IT 外企？大专生能进 IT 外企吗？简历怎么写才标准？面试题范围是什么？面试中需要注意哪些方面？

诸如此类的很多问题，在本篇中，基于我们多年在业内的“摸爬滚打”，我们将给出回答。

本篇主要介绍软件测试业内的一些概况、简历与面试要点以及面试笔试题出题范围等，以供大家参考。

第1章 软件测试面试概述

1.1 软件测试有前途吗

相信选择翻开这本书的读者当中一定有部分读者正在对自己的职业道路进行选择。作为计算机相关专业的毕业生或从业人员，软件测试可能就是其可选项之一。那么我们不禁会问：为什么要选择软件测试？软件测试的工作有前途吗？

测试行业由来已久，在传统制造行业中它一直是用来保证产品质量的一个重要手段，直到现在仍然蓬勃发展，越来越多的新技术被运用到测试行业当中。随着人们意识到品牌的力量，以及产品的高质量对品牌的贡献，人们对质量检测和质量保证的投入越来越大。

软件制造业是制造工业中的一员，是传统制造行业的延伸与发展，其发展到今天同样经历了与其他制造行业的从无到有、从稚嫩走向成熟的过程。

在计算机诞生的初期，编程语言的落后及程序的简单，使得软件的测试仅仅停留在一个程序调试的概念内。此时只要简单的程序能够正常运行即可，并不需要什么专职的测试人员。

之后随着一些结构化的高级语言的出现，如 FORTRAN、PASCAL、C 等，程序变得越来越复杂，软件的规模也随之变得越来越大。软件测试的重要性也就逐渐显现出来。专职的测试人员也开始逐步走上这个舞台。

PC 及 C++等面向对象编程语言的出现使得软件的规模产生了爆炸式的增长。动辄几十万、几百万行代码的软件产品给软件测试行业带来空前契机。各种软件测试理论语后春笋般地出现。这些先行者的经验和理论也就成为后来人的宝贵财富。当然这个时期的软件测试还是以手工测试为主。

之后 PC 和互联网的普及以及多媒体技术的成熟和大规模的使用使得软件测试提高到了一个新的高度。这时的软件测试不仅仅是单纯的测试软件，还要求对产品的缺陷趋势进行评估，还要对软件产品的测试和质量风险进行评估。同时软件的测试工作也开始牢牢地渗透进了软件生产流程中的每个角落。随着软件产品规模的扩大，为了降低软件测试的成本，出现了一大批自动化测试工具。这些工具有的直接帮助测试人员开发（或录制）、执行测试用例；有的帮助测

Offer

试人员管理测试测试用例、跟踪软件缺陷；有的帮助项目管理人员和测试人员分析测试结果，预测产品的缺陷数量走势。

软件生产的工业时代来临了吗？从现阶段来看，软件产品的规模在日益扩大，但人均生产率却在逐步降低。进入了工业时代应当以大量机器和自动化手段代替人工生产，大规模提高人均生产力为标志。而现今软件的生产还是主要依靠大量开发人员的手工编码和测试人员的手工测试，远没有实现软件的工业时代。

因此，软件行业留给我们测试人员的机会还很多，例如创造更多有价值的理论、提高自动化测试工具的普及、从理论上提高测试用例自动化执行的覆盖面和发现软件缺陷的频率等（现有的自动化测试用例通常仅被提倡用于冒烟测试、回归测试和版本验证测试等，其发现软件缺陷的能力也有限，仅为总量的20%~30%）。这些对于软件测试人来说任重而道远。

总的来说，软件测试行业在软件工业中的地位逐步提高，趋势不减。测试人员在软件开发项目中所占的比重也越来越大。同时对软件测试人员本身的素质要求也越来越高。当初只具备简单的计算机基础和部分软件产品业务逻辑的软件测试人员现今已不能满足大规模、有计划、讲策略的软件测试活动的需求。取而代之的是既懂理论，又重实践；既懂开发，又懂测试；既有丰富基层的测试经验，又有组织管理测试技巧的复合型人才，这样的人才也就自然而然地会得到一份丰厚的回报。所以，如果你刚刚开始选择或已经选择了软件测试这个行业，你会发现前面有一片广阔的天地等你去闯。你可以将本书作为一份对自己测试技能的核对清单，逐步提高清单上的各项技能，从而成为猎头眼中的“猎物”，老板们眼中的关键人物。

1.2 测试比开发水平差吗

在国内，一种带有偏见的观点很有市场，即测试工程师比开发工程师水平差，真是这样的吗？

对于任何行业，从业者水平分布都是金字塔形。

1.2.1 初级测试工程师

很多人理解软件测试就是在计算机前按来按去寻找软件缺陷。诚然，初级软件测试工程师一般只做黑盒测试，也就是说，按部就班地执行测试用例，当预期结果与实际结果不符时，就认为可能寻找到一个软件缺陷。他们往往不关心缺陷后面的逻辑结构。

初级测试工程师往往刚毕业缺乏经验，或是经过短暂的职业培训初窍门径，很多人甚至不是工科专业毕业。

这种岗位不可能是企业的核心岗位，因此很多大型公司都把这种岗位外包出去，由外包公司派来劳务派遣人员来从事此种工作。



初级测试工程师基本没有开发经验，编程能力欠缺，很可能读不懂程序。例如，当出现两个软件缺陷现象不同，但根源相同的情况，由于缺乏对程序逻辑的背景知识，初级测试工程师往往就无法区分，这样就造成相当数量的重复报告，增加了开发人员重复劳动，造成了缺陷统计数据不准确。

1.2.2 中级测试工程师

中级测试工程师可能不如开发工程师在某个领域深入，但是知识面要宽一些。

中级测试工程师通常具有一定的开发经验，而且对质量管理、网络、多种操作系统、多种脚本语言、数据库等领域都具有相当程度的知识。另外，中级测试工程师往往较普通开发人员对项目架构接口等细节有更深的了解，他们需要纵观全局并且注重细节才能设计出有效精彩的测试用例，经常需要见人所未见，还会对需求提出改进意见。

由于对项目了解是入木三分，中级测试工程师往往还会负责一些涉及全局性的工作，例如写产品说明书以及向客户提供技术支持等，即使遇到公司裁员，这样的测试工程师也会被保留，普通的开发工程师可以再招聘，反正他们只了解软件的一小部分。而掌握细节和全局的测试工程师若被解聘，很多重要内容可能会丢失，项目失败风险大大增加。

1.2.3 高级测试工程师

高级测试工程师需要具有丰富的开发知识和经验，既懂开发又懂测试，是复合型的通才，因为高级的白盒测试、性能测试、测试开发及安全测试等都需要强大的编程能力，高级的安全测试甚至还需要出色的反汇编功底。这些高级测试工程师在国内数量非常少，年薪也比较高。通常只有大公司才请得起这样的高薪人才。

高级测试工程师的工作和开发结合得非常紧密，他们大多是由开发工程师成功转型而来，深刻理解开发领域与测试领域，正所谓是：“站得更高看得更远”。

想成为高级测试工程师，需要在技术领域深加钻研，需要学好计算机专业课程，需要进行大量的软件开发实践，还需要刻苦钻研软件测试知识。由于在我国的教学体系中涉及这些领域甚少，因此每前进一步，完全要依靠自己钻研，需要付出大量心血。

当达到高级测试工程师的程度时，专业权威之路不再是遥不可及。

1.3 软件测试工程师能挣多少钱

软件测试工程师未必都是低薪。对于一线工程师，工资分布从月薪 2000 元到年薪二三十万元不等。



如果是外企大公司的正式员工（注：不是外包员工），则发 13 或 14 个月的薪水，同时还有数万元的年终奖。

下面来介绍一下软件测试工程师的工资分布情况。

1.3.1 黑盒测试

对于黑盒测试工程师，薪水从月薪 2000 元到 10000 元不等。大公司给的高一些，对于大的外企公司，即使处于外包岗位，该职位员工月薪也可能达到 7000~8500 元。

1.3.2 测试开发

测试和开发是软件工程师的两大类。从更深层次来看，测试中包含开发，开发中也包含测试。在交付给测试人员之前，开发人员会测试自己的程序。而测试人员经常需要开发一些小工具或程序来辅助测试。例如 XUNIT 系列单元测试工具需要编程，测试人员还需要写一些工具如 LoadRunner 的执行脚本。

开发用于测试的工具，无疑是测试工程师中薪水最高的领域之一。这需要深层次的开发知识和经验，同时还需要对测试有着深刻的了解。在国内这种职位的薪水，保守估计也应当在月薪 10000 元左右。这样的工程师通常精通 C++、Java、.NET、C#中的至少一种，或者 UNIX Shell、Perl、Pathon、Ruby 中的至少一种。

1.3.3 性能测试

性能测试是另一个高薪领域。高水平的性能测试工程师通常精通 LoadRunner、Jmeter 等性能测试工具，拥有深厚的网络知识及数据库知识，同时对系统的架构有着深刻理解。这种职位的工程师需要具备较深的开发背景，薪水保守估计应在月薪 10000 元左右。

1.3.4 白盒测试

白盒测试工程师在国内比较少见，其薪水较高，一般公司都是由开发工程师在开发阶段来做白盒测试。笔者于 2008 年末曾得到过国内某著名公司的面试机会，当时该公司提供的月薪是 14000 元。通常白盒测试工程师要精通 XUNIT 系列单元测试软件的一种或几种，对所测试的开发语言有着丰富的经验，他们开发的水平甚至要超过开发人员。

1.3.5 安全测试

高级安全测试工程师的薪水也相当高，至少也在月薪 10000 元左右。

优秀的计算机基础和动手实践能力是高级安全测试工程师的必备能力。有位专家曾说过，不懂二进制数据就无从谈起安全技术。



安全测试工程师可能精通反汇编，对 XSS（跨站脚本攻击）等攻击能力得心应手，本身就是个优秀的黑客，擅长寻找安全漏洞。这个工作需要精通汇编语言和网络知识，对计算机语言的理解达到比普通开发更深的层次。

安全测试工程师精通黑客技术，例如精通汇编语言和反汇编技术，精通二进制机器代码，精通密码学知识，了解 PKI 体系架构，擅于分析病毒样本。他们有能力拿掉 PE 上复杂的保护壳，并且在二进制文件中定位漏洞。

至于中小公司的安全测试工程师，他们的薪水可能不会特别高。

1.4 职业培训有用吗

职业培训非常有用，尤其对有一定工作经验的人更有用。

一些有工作经验的测试工程师，往往只凭经验来测试，从而造成知其然而不知其所以然。他们迫切需要理论指导实践。在学习中，这些人能清醒地发现重点，更好地安排学习，将主要精力按照八二原则分配到重点知识领域。

而对于一些没有工作经验的学员，在学习中是有些盲目的。由于没有切身经验的印证，也没有重点与非重点的概念，不得不“眉毛胡子一把抓”，很可能在非重点领域不必要地耗费大量精力。

相比之下，有工作经验的学员，通常学习动力比较饱满，而且在社会上经历过锤炼，会倍加珍惜学习提高的机会。另外，这样的学员通常花的是自己的钱，不是父母的钱，也会努力进取，对得起自己的辛苦钱。

实际上，职业培训可以帮助我们举一，而我们自己可以进行反三的提升。在课堂内容之外，我们如果进行大量的学习，会有更大进步。

在我国，很少有高校开设软件测试这个专业，即使在一些高校中开设了此专业，软件测试专业课程的设置也不尽合理，往往是重理论，但不够贴近实用。另外，同其他专业一样，在高校的软件测试专业中，学生几乎没有什么机会来训练动手能力。

对于那些想进入软件测试领域的非计算机及相关专业的人而言，职业培训是一种非常难得的入门途径。通过职业培训，他们可以学习到软件测试的专业知识，而且在履历上可以让软件公司的 HR 看到了与测试工作相关的经历。

对于没有什么工作经验的学生而言，职业培训不但提供了动手实践的机会，更重要的是指出了这个领域内的各种各样的发展方向，学生在培训中，有途径了解到很多圈内的规则、潜规则、好消息、坏消息等重要的信息，不再是完全的职场生手。另一方面，通过职业培训还可以使得他们了解如何将学校的知识更好地应用到实践，并达到一个更深层次的认识。



1.4.1 高校教育与职业培训

对于软件测试这个专业，在国内各重点高校中，只有极少数几所高校开设，而且多是在硕士研究生阶段，本科阶段几乎没有设置。

说明

有一些专科院校虽然开设了软件测试专业，但是教材师资非常欠缺，常常要请职业培训机构的老师去上课。

笔者曾在某著名职业培训学校参加过培训，获益良多。其课程设置贴近实用，教材作者都是一线的测试工程师，具有丰富的实践经验。在教学中，该培训学校安排了大量的动手实践机会，而且最后还会安排一个实战，与企业合作，由老师带队，测试实际的项目。

1.4.2 为非专业人士和学生打开一道门

低端的软件测试工程师一般不需要深厚的计算机专业背景，诸如离散数学、数据结构这种计算机专业基础课根本不需要学习就可以上岗，唯一需要的只是一段时间的测试基础培训。职业培训提供了这个必不可少的环节，降低了用人单位的培训成本。不论何种专业背景，只需大专学历即可。

1.4.3 有工作经验者获益最大

有经验往往会占据优势。

职业培训也是如此，有工作经验者占优势。带着问题去学习，理论立即又可以用来指导实践。

我有切身体会。去培训之前，我已做过一段时间的测试工程师。在上课时经常发现，在工作中遇到的许多实际问题，老师会提到并讲解，于是我赶紧做笔记，如获至宝。而此时那些没工作经验的同学，往往茫茫然不知是重点。

警示

在工作中我发现，很多测试工程师，甚至很多资深测试工程师，往往是凭工作经验来测试，欠缺理论体系的支撑。实践需要理论的指导。

职业培训可以帮助有经验的测试工程师搭建理论体系，知其所以然，找到方向，找到不足。例如在设计测试用例时，根据分类理论选择好测试模型，而不是天马行空过于发散，仍然还会遗漏重点。

在培训完之后，既有工作经验，又具备理论体系，在工作方面可以有个巨大飞跃。

引申

在工作中，必须坚持不断自学，正所谓是，逆水行舟，不进则退。



1.5 大专生能进IT外企吗

大专生完全可以进入IT外企，但进入著名的大公司比较难。如果非常出色或是工作经验丰富，也有可能进入大公司，但可能多是以外包人员的身份进入，转正比较难。

1.5.1 市场概况

对于较低端的测试工程师，目前市场上供大于求。由于低端的测试工程师基本上只从事黑盒测试中简单的部分，对专业和学历的要求并不是很高，所以很多IT外企愿意使用大专生。

一般来说，大专生进入IT外企时，不论是作为中小公司的正式员工，还是作为大公司的外包员工，月薪范围大致是1500～4000元。

随着工作经验的增加，工资也将会有相应提高。如果是一线的测试工程师，如果在专业水平或学历方面没有大的提高，估计最高能达到月薪6000元左右。

1.5.2 参加职业培训增强竞争力

由于IT外企招聘大专生的主要目的是节省成本，所以建议应聘人员要去参加一下软件测试的职业培训。

例子

笔者在工作后参加过软件测试工程师培训班，深感受益良多。有很多没有工作的大专生在那里学习，经过职业培训后，这些大专生可以立即上手黑盒测试工程师的工作，使得IT外企节省了培训新员工的大量成本，因此这些学员都有一定的竞争力。

其实，不论选择哪个职业培训机构，加强实力是找到工作的最重要筹码。能加强大专生的软件测试专业水平和实践能力，是职业培训的重大意义。大专生在学历上没有优势，如果没有一定的专业优势，是很难和本科生竞争的。

所以在目前这个阶段，由于科班出身的人基本没有，软件测试职业培训机构对大专生的培训是非常有价值的。

1.5.3 进入IT外企所需的基本条件

IT外企在招聘大专生时，所要求的基本条件通常是。

（1）具有一定的软件测试专业水平，最好能马上上手干活。

（2）英语通过大学英语四级。能看懂英文工作文档（速度不能过慢），可以用英文回复电子邮件，大致正确（尽管也许不精确）地表达自己的意见。

（3）专业上没有限制一定要是计算机或相关专业。某职业培训机构中不少大专生都是非工



科专业的，但找到工作的比比皆是。不过相比之下在招聘时，计算机及相关专业要占些优势，工科比非工科要占优势。

1.5.4 如何提升自己

只拥有大专学历是不够的，在 IT 企业里，高学历的优秀人才比比皆是。

例子

记得我高考那几年，成绩最好的学生基本上选择的都是计算机、通信或电子专业。

大专生需要利用业余时间多学习来提升自己，同时还需要在专业领域努力进取。软件测试有很多领域，钻研进去，收入必然会水涨船高。

另外，大专生还需要在英文方面多下工夫，这样才能在外企中更好地生存和发展。

1.6 不擅长不喜欢开发就适合做测试

问到选择软件测试作为职业的原因，有些测试工程师会回答说因为不擅长或不喜欢开发的工作。

这样的说法割裂了测试和开发的紧密联系，通常反映了开发能力的缺乏（也或许是对开发工作的厌倦）。事实上，测试和开发是“你中有我，我中有你”。本书内容的重点之一就是测试中的开发。诚然，对于黑盒测试中有些浅显的测试，并不需要多少开发知识，可是薪水也是测试中最低级的。事实告诉我们，拥有出色的开发能力并努力将之应用于测试中的测试工程师，竞争力更强，薪水也会位于软件测试业的顶级。

1.6.1 判定缺陷间的重复及依赖关系需要开发能力

很多时候，两个软件缺陷表现形式不同，但是根源相同，这样就形成了重复缺陷；或者有时，一个缺陷基于另一个缺陷，当另一个缺陷被解决了，这个缺陷就会消失，这属于缺陷间的依赖。

我们知道，有效缺陷的数量会用于统计项目的情况，重复或依赖性的缺陷数量达到一定程度时，势必影响领导对于整个项目进展的判断，进而带来很多的风险，因此各个测试部门都要求尽量减少这种无效缺陷的提交。下面举例说明，如图 1-1 和图 1-2 所示。

粗看起来，这似乎是两个不同的缺陷。可是有开发能力的人会知道，这二者的解法实则相同。只要限制日期文本框即只从日期控件中选择而不可手工填入，这两个缺陷就都被解决了。由于解决方法相同，因此这两个缺陷应当被判定为重复。

有一定开发经验的测试人员，对于缺陷间的重复或依赖，在从前的开发实践中可能就遇到过类似情况，而更可以举一反三，进行更精确的判断。而这种判断，对于无开发能力和经验不

Offer

足的人来说，是很难达到的。

例子

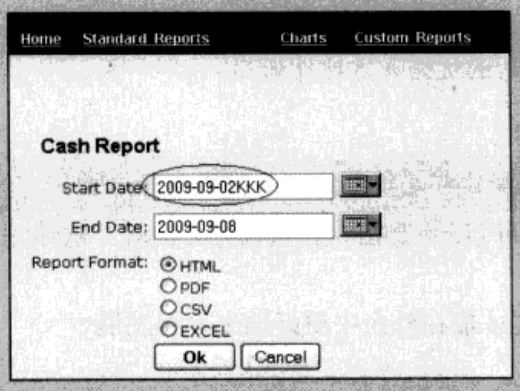


图 1-1 Bug1-1

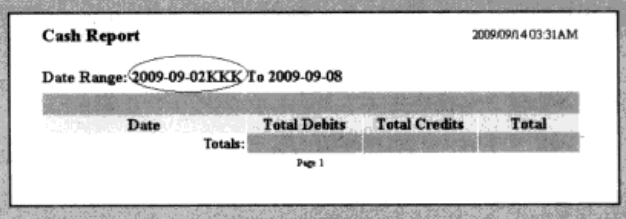


图 1-2 Bug2-2

Bug1：当在日期控件中选择一个日期后，在该日期后可写入任意字符，如“KKK”，单击 OK 按钮生成报告，报告中的日期含有“KKK”，是不合法的日期。

Bug2：当在日期控件中选择一个日期后，在该日期后输入 3000 个字符后，单击 OK 按钮，程序崩溃。

1.6.2 使用自动化测试工具需要开发能力

无论是著名的性能测试工具 LoadRunner、Jmeter，还是摩托罗拉的手机测试工具 FlexAuto，很多工具的使用都需要编写脚本程序。

说明

脚本程序既没有 C++或 Java 语言那么难，也不是面向对象的，只是简单的解释型语言。

拥有 C++或 Java 等复杂语言的开发能力能帮助你更好地理解脚本语言，出问题时能够更快速、更准确地定位问题出在哪里。丰富的开发经验能使你事半功倍，得心应手。

欠缺开发经验的人，即使在编写脚本语言中的简单语言时，也不免错误百出。因为缺少那种开发的思维，既不能规划好全局，也不能照顾到细节，这样的项目，失败的风险大大增加。

1.6.3 黑盒测试偏爱开发能力

在黑盒测试中虽然看不到代码，但是如果有开发能力，则测试人员将可以更容易贴近开发人员的思路，从而更好地设计和执行测试。

说明

测试人员经常会琢磨：“如果我是开发人员，在开发这个模块时可能会犯什么错误？”

Offer

按照需求逐条测试，只能保证需求上写的部分是正确的，而其他很多部分是覆盖不到的。



根据 80-20 原则，80%的测试要用于 20%高风险的地方。同样，每个人被分配的任务中，也有 80%与 20%之分。如何来判断哪些是 80%的呢？事实证明，拥有一定的开发能力将帮助你准确地判断，从而使我们更聪明、更积极、更有效地完成工作。

1.6.4 白盒测试需要开发能力

例子

在某应用中，有个生成网站统计报告的新项目进入了系统测试阶段，这个项目的核心部分是在数据库里运用 SQL 查询结果。领导布置了一个任务，要求我自己来重写一遍所有的 SQL 查询，来对比一下报告数据。

首先我需要看懂代码。如果粘贴到 Word 文档上，每个查询语句平均有 80 行。这条的报告一共有约 40 种。

当看懂代码且理解了逻辑之后，我开始试图找出其中的 Bug。一些典型的 Bug 有。

（1）边界值错误。通常发生于“between, and”语句。

（2）限定条件错误。例如在 where 子句中少写了一个 and 条件。

（3）有的表无用。例如 left join 了个表，发现这个表没有用到或一些字段与另一表中的字段重复。

（4）函数使用错误。

（5）字段或表的混淆。例如应使用 terminated_id 字段的地方误用了 end_id 字段。

说明

通常意义上的白盒测试不需要重写开发人员的代码，只需要理解并测试其逻辑即可。

最后我参照开发人员的代码来重写我的查询，并用其在数据库中运行的结果来对比软件生成的报告。

大家可以看到，在整个测试过程中测试人员的确需要一定的开发能力。

1.6.5 安全测试需要开发能力

在进行安全测试时，作为测试人员，经常需要模拟攻击系统。

没有开发能力，就不知道可能的薄弱环节在哪里。

以一些简单的命令来攻击，这个层次比较浅，无法模拟出真正的高手来攻击的态势，也就是说，漏洞可能很严重，但是我们的测试水平达不到，根本测试不出来。

安全测试是一个计算机学科中很深的领域，很多高手不但精通多门开门语言，而且精通反汇编，对计算机运行原理了如指掌。

Offer

1.6.6 开发测试工具

为了实现对某些项目的测试，部门中经常会要求开发一些小测试工具。例如摩托罗拉公司的手机部门，需要开发一个工具，来自动化集成所有代码生成软件的一个版本，然后自动分发给相关部分，并进行一些基本的通过性测试（冒烟测试）。

这需要很强的开发能力。如果你有这种能力，显然就可以超越其他同事，成为部门的技术骨干。

本书的作者之一——于春智，就曾经用 Perl 语言开发了这样一个工具，使得整个部门的工作效率大大提高。例如，原先需要 10 个人手工集成要一整天的时间，但通过于春智开发的工具，仅需要几分钟就可以完成自动化集成。

1.7 有效沟通

是否经历过冗长无目的、无意义的会议？

是否经历过在会议上无从插嘴？

是否经历过开完会后全然不知未来任务？

是否经历过因为要求更正一个缺陷而与开发人员之间的唇枪舌战？

是否会遇上对方出尔反尔但自己又苦于没有证据来证明你们事先达成的一致？

是否遇上过老板口头交待的事情，时间长了竟然发现有几件事情压根被遗忘？

以上的情况不但消耗了我们的时间，还消耗了我们的精力和耐力。而以上的所有这些都可归结为沟通的问题。如何提高沟通的效率以及如何减少沟通产生的内耗是本节的焦点。

1.7.1 沟通方式之思辨

下面先来看看不同的沟通方式有什么优势和劣势。

- 面对面：最为直接和有效的即时沟通方式。沟通时可以直接看到对方表情，感觉对方语气的变化，容易感觉到对方对你所陈述的事情的感兴趣程度和接受程度。该沟通方式效率最高，可一对多或多对多的进行，但存在对沟通各方距离的限制。
- 电话：非常有效的即时沟通方式。沟通时可以感觉对方语气的变化，较容易感觉到对方对你所陈述的事情的感兴趣程度和接受程度。该沟通方式效率很高，可一对多或多对多的进行，但通常用于一对一的沟通。该方式突破沟通各方距离的限制，但会产生额外的费用。
- 即时通软件：较为有效的即时沟通方式。沟通时只能通过字里行间隐含的信息感觉对方对你所陈述的事情的感兴趣程度和接受程度。但有时对方即使在线也可以假装不在线，或沟通时以怠慢的方式进行对话，对话的结果可以保存，并用来作为对某件事情做出决

Offer!

定的证据。该沟通方式效率较高，可一对多或多对多的进行，但通常用于一对一的沟通。该方式无距离的限制，即使沟通各方跨越国界也不会产生任何费用，而且能够记录沟通的内容。

- 邮件：最为正式的一种沟通方式，但不即时。对方在沟通时可以怠慢或拖延的方式进行回复，甚至不回复。该沟通方式效率较低，但是在多对多沟通形式下更为有效。该方式无距离的限制，即使沟通各方跨越国界也不会产生任何费用，而且能够准确、正式地记录沟通的内容。

尽可能以最高效的方式进行前期的沟通，然后以邮件等手段对做出决定进行确认和归档。例如老板口头交代的事情，就可以以邮件的形式加以确认，这样既加深了自己的映像又给自己做好了一张核查列表，以备日后跟踪完成情况。

下面介绍其他因组织形式不同而产生的变体。

- 会议：面对面沟通方式在组织形式上的扩展。有效的会议一般要求组织者在开会之前就将会议的目的和各项议题的安排以书面的形式发给其他与会者。与会者对会议的内容做好准备，对与自身相关的内容做好发言的准备。在会议进行过程中最容易出现因某项议题各方争执不下的局面，这时组织者可以事先规定某项议题的讨论时限。如果在规定时间内没有完成，则将议题的讨论现状记录下来，并责成相关人等在会下进行再调研、再讨论，同时要求相关人等给出能够做出最后讨论结果（不管是否能达成一致）的时限。对于能够顺利通过讨论的议题，会议组织者应将讨论的结果一一记录。如果需要其他人根据相关结果在会后做进一步工作的，则一定要责任到人，并要求其对完成时限做出承诺。在会议结束之后会议组织者应当将会议的议题、各项议题的讨论结果和重要发言的摘要以及后续工作的责任人和其所承诺的完成时间写成会议记录，而后发送给会议参与者和其他相关人等。
- 电话会议：是会议在物理空间上的扩展。通常同网络会议（NetMeeting）软件配合使用。因为电话会议的参与者大多数情况下可能都看不到对方，所以对会议组织者的要求更高，同时要保持会议的连贯性，以防止冷场。在会议开始前也要对电话网络和互联网络进行必要的检查，以免在会议开始时无法将与会者正常接入，从而耽误会议的进程。除此之外，就可以按照普通会议的流程进行。

1.7.2 When and How

在测试的各个时期，测试人员都需要与项目中的其他角色进行大量的沟通。测试人员与测试小组外部角色进行沟通，通常发生在以下阶段。

（1）文档审查时期。

在文档的个人预审阶段应仔细阅读文档，并将文档中的问题、缺陷和不清楚的地方记录下来，以备在正式评审会议上提出。在与其他角色人员讨论文档时，测试人员要始终站在客户的角度看待文档中的问题，多考虑客户会怎么想。当与对方发生争执时，应当站在客户的角度思



考问题并以此引导对方。除此之外，文档审查的一个重要目的就是要让参与审查的各方对文档有一致的认识。不要将文档中不确定的描述遗留到后面的阶段，这有可能会为以后的争执埋下伏笔。在评审的会议上要采取灵活的方式指出问题。提出观点时要针对问题而不针对个人。在各方意见达成口头一致后，评审会议的组织者应当将对文档的修改和讨论结果记录下来，并责成责任人按时完成对文档的修改，然后将其发送给会议参与者和其他相关人等。

（2）测试计划和测试用例审查时期。

这两个时期里测试人员通常是文档的作者和审查会议的组织者，因此为了事先与软件项目中其他各成员进行更加有效的沟通，文档作者应当在评审会议前至少一周将文档通过邮件等正式的形式发送给参与评审的角色，以便各角色能够对文档进行阅读，发现错误，并在文档发出到评审会议开始的这段时间内定期地提醒参与评审者提交发现的问题，以及文档中写得不够清楚的地方。对待他人提出的问题应当正确对待，不要把别人提出文档的问题看作是在挑刺。在收集完问题之后，要站在客户的角度积极地对这些问题做准备，在评审会议上要对前期收集的问题一一做出答复。遇上会议上提出的新问题应当仔细分析，详细解答。如果会议上不能立即解决的问题应当将其记录下来。在评审会议结束之后要将会议中解决的问题、达成的一致和为解决的问题进行归档，并安排责任人对其进行跟踪。然后将以上归档和责任人的安排制成会议记录，再将会议的记录发送给与会者和其他相关人等。

（3）发现软件缺陷或跟踪缺陷时期。

这个阶段测试人员的沟通对象主要是程序的开发人员。而测试人员在发现缺陷时经常要向开发人员咨询以确认这个缺陷。这时，我们在进行确认之前要确定是否已收集了以下信息：软件缺陷可能对客户产生的影响，重现的步骤，精确定位了软件缺陷的重现点，以及测定软件缺陷的重现概率。然后将这些信息传递给开发人员，并听取开发人员的反馈，以确定这是否是一个真正的缺陷。当双方意见不能达到一致时，可以征求更大范围的技术人员意见或提交到上一级管理人员进行讨论。之后，如果测试人员认定这是一个软件缺陷，则将其报告为一个正式的软件缺陷，记录在缺陷跟踪系统中。当然，有些测试人员可能会在确认之前就将其记录在缺陷跟踪系统中，这样做的后果很可能是该测试人员所报告的缺陷质量不高，时而不能重现，时而被开发人员以各种理由退回。这样都有可能影响测试人员的声誉。所以测试人员在报告缺陷之前应该先与相关人员进行必要的沟通。

（4）其他。

当然，在软件测试活动的其他阶段，也伴随着大量的与其他工作人员的沟通。测试人员应当站在客户角度思考的行为习惯引导自己和他人。在所作的决定影响到他人时，应当充分收集信息，做好准备工作，并在此基础上再进行大量的沟通，最好在达成一致之后再执行先前所作的决定。

第2章 软件测试面试前的准备

2.1 简历要点

简历在应聘中的重要作用是不言而喻的。HR（人力资源）通过筛选简历来寻找合适的应聘者。如何在众多的简历中被一眼选中，至少不被淘汰，是很值得探讨的。

有的人很优秀，但是简历写得不上档次，仍旧很有可能被淘汰掉。有的人很会推销自己，即使不是牛人，仍旧会得到面试机会。

那么 HR 究竟想看什么呢？

求职意向、薪资要求、教育背景、工作年限及工作经验等无疑是重中之重。因此，我们就要把这些重点写在前面。

除了个人简况（如姓名、出生年月日等）也写在前面之外，诸如论文发表、获奖情况、培训经历等都要放在这些重点的后面，不可喧宾夺主。

简介需要开门见山，例如想找什么工作、想要多少钱，一目了然。这帮助 HR 节省了时间，获得 HR 初步的好感。

简历不讲究文采，不能写成文言文，一定要清晰、客观、无歧义。简历的格式有一定之规，不要在格式上突出个性，基本没有意义，关键是有实力、有内容。

正是由于简历的重要性，简历才会被反复修改，通常能达到 30 遍以上。

本节主要介绍简历中的注意事项，并给出一个格式模板。

2.1.1 注意事项

- 求职意向和薪资要求应当放在最前面

有些应聘者，尤其是学生，不知道自己适合做什么工作，因此不写求职意向。这些应聘者期盼 HR 为之选择。可是实际上这可能吗？HR 是没有义务帮你选一份工作的。他们也不可能比你更了解你自己。所以求职意向一定要写得具体。



如果是刚毕业的学生，对薪水没有什么要求，薪资要求一栏中可以写“面议”。

- 工作年限也要放在醒目位置

有几年工作经验是 HR 衡量应聘者水平的重要标准之一，因此要放在紧跟求职意向和薪资要求的地方。

- 教育背景放在工作经验之前

这样安排是由于时间顺序，多数人都是先上学再工作，所以 HR 思考的习惯是要先看学校再看工作经验。

教育背景按逆序排，即最新的教育经历放在最前面。

强调

在整份简历中，涉及时间的部分都是逆序排列。这样做是因为更近的经历反映你最高的水平，也更接近你目前真实的情况。

- 教育背景中几乎没有必要写中学阶段

在教育背景中只需列出读大学之后的教育经历即可。中学阶段没有学专业，列上中学意义不大。

引申

在整份简历中，如无特别必要（获奥林匹克奖还是可以炫耀一下的），没有必要提及中学阶段。

- 学生时代的项目经验可以写成工作经验

如果是刚毕业的学生，则可以在工作经验一栏中填入学生时代的项目经验。

如果已工作过，学生时代的项目经验可以不写。

- 工作经验要有层次条理

每一份工作经验都应按照单位名称、项目名称、职务、工作内容及工作业绩的顺序来组织。

工作内容和工作业绩如果不好分开，也可以写在一起。如果没有什么工作业绩，则可以把工作业绩省略。同时应尽量列出具体的数据，避免模糊的描述。

例子

好的描述：业绩全部门第 1 名，平均每月提交有效 Bugs120 个。
不好的描述：工作异常出色被大家称道。

先写最重要的工作内容和最好的工作业绩。

- 莫写自我评价

对自己的认识很难是客观的，再妄图用自己对自己的主观评价来影响 HR 的判断，这本身

就是不合逻辑的。HR 不喜欢这种非客观的东西。HR 判断你的水平是通过你简历中的各种经历来判断的。这种自我评价是学生时代填各种档案表格遗留下来的习惯，在写应聘简历时可以抛弃了。只有摆事实、用数据，才更有说服力。

- 所有不相关所应聘工作的内容一概不要写

有很多人好大喜功，把一大堆东西罗列上简历，也不管和所应聘的工作相关与否。这些应聘者以为 HR 会对自己的多种能力很感兴趣。实际上，HR 根本不会考查你其他领域是否优秀，他们要找的只是适合于这份工作的人。写一堆杂七杂八的事，除了让 HR 感觉不着边际、啰嗦、遮蔽重点之外，没有什么好处。当然，如果你要应聘高管，多种能力可能会使 HR 感兴趣。对于一个普通员工来说，关键是能否胜任本职工作。

例子

例如，笔者在大学时是校篮球队的，也是学院记者团特约撰稿人，但在笔者简历中根本就不会提及这些，因为这些和软件测试工程师的本职工作没有关系。

- 简历内容可超过一页纸

咨询了西门子、微软和汤森路透的 3 位资深 HR，她们一致认为“简历内容不可超过一页 A4 纸”这种说法是错误的。

只要是与工作相关内容，超出一页无妨。工作经验丰富、项目经验丰富、专业方面获奖丰富，这都是给应聘者加分的事情，怎能拘泥不写？

当然，语句宜精练。

- 时间不能有空白

在有的简历中，毕业后有一段时间是空白的，HR 不免会有丰富的联想。因此，即使这段时间你是赋闲在家学习，也应当写清楚。

- 相片可以不要

由于软件工程师对相貌要求不高，所以简历中可以不放相片。当然，放相片也没有坏处。相片可以放在简历左上角或右上角的位置。

- 从强到弱排列专业技能

在专业技能部分，应先写“精通”的专业技能，再写“熟练”、“熟悉”等。因为前面几行比较醒目。

- 薪水增幅通常最高为 30%

猎头常常说，在 IT 行业，跳槽后工资增幅最高通常为原先的 30%。除非你非常优秀。因此在写期望薪水时，自己心里要有杆秤。

- 不要编造

说一个谎，就要十个谎言来圆谎。说谎的成本太大。



2.1.2 简历格式模板

个人简历

姓名：×××　　学历：工学硕士　　专业：软件工程
应聘岗位：高级软件测试工程师　　期望薪水：16000元/月
目前薪水：12000元/月　　工作经验：6年　　出生年月：1979.7　　性别：男
硕士毕业院校：英国帝国理工大学计算机学院，2002.9~2005.10
本科毕业院校：南京大学计算机学院，1998.9~2002.7
联系方式：×××××　　电邮地址：×××××

工作经历：

公司名称：XXX（中国）电子有限公司
项目名称：×××××
职位：×××××
工作时间：2007.2至今
工作内容：1．×××××
　　2．×××××
　　3．×××××
工作业绩：1．×××××
　　2．×××××
　　3．×××××

公司名称：微软亚洲研究院
项目名称：×××××
职　　位：高级软件测试工程师
工作时间：2006.2~2007.2
工作内容：1．×××××
　　2．×××××
　　3．×××××
工作业绩：1．×××××
　　2．×××××
　　3．×××××



公司名称：谷歌信息技术（中国）有限公司
项目名称：×××××
职　　位：开发/测试工程师
工作时间：2005.10~2006.2
工作内容：1．×××××
　　2．×××××
　　3．×××××
工作业绩：1．×××××
　　2．×××××

专业论著：

1．人民邮电出版社，《×××××》，第一作者。
2．人民邮电出版社，《×××××》，第二作者。
3．专业论文《×××××》，×××杂志。第一作者。2009.9。SCI 收录。
4．专业论文《×××××》，×××杂志。第一作者。2009.6。SCI 收录。
5．专业论文《×××××》，×××杂志。第三作者。2009.2。SCI 收录。
6．专业论文《×××××》，×××会议。第一作者。2008.12。EI 收录。

专业技能：

1．精通×××××。
2．精通×××××。
3．熟练的×××××编程。
4．熟悉×××××理论。

英文能力：

1．TOEFL：640。GRE：2320。CET-6：92（级别：优秀）。词汇量 5 万以上。
2．2007.9~2008.10，在国家一级杂志《×××××》和《×××××》发表英文翻译文章 11 篇。

培训经历及专业证书

2008.2~2009.2　×××××
2006.5~2007.5　×××××



2.2 面试要点

当简历被 HR 选中之后，应聘者将会得到面试机会。

正式员工的面试流程通常是：电话面试→笔试→面试。

外包工程师的面试流程是：外包公司电话面试→外包公司笔试→外包公司面试→所派驻公司笔试→所派驻公司面试。

对于外包工程师的面试，外包公司本身的电话面试、笔试及面试基本上只是走个过场，绝大部分人都能通过，关键在于所派驻公司的笔试和面试。

下面我们分别说明电话面试、笔试及面试的情况和一些注意事项。

2.2.1 电话面试

电话面试主要是考查应聘人员基本的英文口语和听力，同时也考查基本的技术背景，时间一般在半小时以内。通常 HR 的英文水平也不会很高，而且一般会提前通知你要电话面试，所以不要紧张，在之前自己和自己说半小时英语，调整自己进入状态即可。

偶尔也有仅凭电话面试就敲定候选人的情况，此时不再进行面对面的真正交流。但通常来说这种职位都是不太重要的职位。

2.2.2 笔试

一旦电话面试通过，即表明应聘人员已具备基本的英文口语和听力能力，技术背景也大致符合招聘岗位，然后应聘者将会被安排来公司进行笔试。

笔试既可能是机考，也可能是用真正的试卷来考。

⚠ 警示

要记得要草稿纸，HR 经常会忘记准备草稿纸。

笔试结束后，需要注意的是，有的公司不让带走试卷，有的公司甚至不让带走草稿纸。

通常笔试的时间是 1~3 小时。有时答到一半，就有人进来收试卷了。因此，时间是个很重要的困素。注意答题时要灵活，不会的先放过去，以先易后难为原则，先答好大部分题。这样即使时间耗尽，经理会发现你大部分基础知识还是掌握得不错，否则会误以为你不懂这些基础知识。

2.2.3 面试

面试是最重要的一关。很多人的笔试成绩非常不错，但是在面试中失败了。

面试通常是由一线主管、HR 来作为面试官，有时还有其他同事。这些人可能一起面试，

也可能分开进行面试。在不能决定选择哪个应聘者时，更高层的领导会参与面试，并最后拍板。

强调

是否录用某人，主要的决定权在于一线主管，而HR只是提供参考意见。

面试官决定录用哪个应聘者，主要是从岗位本身角度考虑的。是否满足岗位需求，也不超出太多（很可能会跳走），即恰如其分，是主要的标准。至于应聘者的形象顺不顺眼，是否是校友，是否是老乡，这都是非常次要的因素。

很多公司的面试，是紧跟在笔试之后的，所以说笔试时间不能耽误太多，否则面试就没有时间了。也有一些公司，如汤森路透，其要求通过笔试的应聘者才有资格进入面试。在这种情况下，笔试和面试往往不是同一天。

对于第一种情况，即使面试者专业上有不足，但仍有力挽狂澜的可能；对于第二种情况，如果专业水平不足，则完全没有任何机会。

通常第一种情况是占大多数的。第二种情况也有优势，应聘者可以回家调整状态，不至于在紧挨着笔试的面试中因过于紧张而发挥失误。

2.2.4 其他注意事项

- 面试官态度和面试结果无关

有时面试官会很凶，也许应聘者感觉其在故意刁难，面试完毕后应聘者往往感觉前途无望；有时面试官和蔼温柔有如春风化雨，面试者如坐春风，以为面试得不错。实际上，面试官态度与面试结果无关。凶恶或和蔼，非常可能仅仅是个人面试风格的不同。有时凶一点，是因为看中了应聘者，给个下马威；有时恶一点，因为已把应聘者看做下属，自己人嘛，要挑剔一下；有时特别客气，是因为没看上应聘者，也不愿意得罪人。

例子

毕业一年后，我去微软公司应聘一个外包的岗位，两个面试官分别面试了我。一位30多岁的男子，彬彬有礼，赵先生长赵先生短，我还以为他对我印象很好，最后侧面了解的结果，他否定了我。另一位20多岁的女面试官，出了好几道难题为难我，横挑鼻子竖挑眼，我当时觉得她肯定看不上我，可是最后她对我非常肯定。

- 临走时要收拾桌子

面试结束，临走时要把水杯带走，不要在桌上留下东西。如果水洒在桌上，则要擦干净，然后把桌椅归位。有的领导比较重视这些细节。

说明

其实我个人认为这些细节无关紧要。不过，我确实听说过有人因此没有被录用的情况。也许你会碰上这样特别严格的领导，因此请大家留意一下，养成随手收拾的生活习惯。

Offer

2.3 考查范围和题型

在IT外企的软件测试工程师的面试笔试中，专查范围和题型分别是什么呢？

2.3.1 考查范围

1．理论基础

这包括两部分，一部分是软件测试概念与原理。例如什么叫回归测试？80-20原则在测试中如何应用？这部分在本书的第二部分测试理论篇的第2章，软件测试理论基础1——概念与原理，进行了讲解。

另一部分是计算机应用，例如C++和Java的开发知识，SQL查询的使用。这部分在本书的第二部分测试理论篇的第3章，软件测试理论基础2——计算机应用，进行了讲解。

2．开放式问题

这部分主要是考查项目经验。例如在项目中最大的成功是什么？最大的失败是什么？假设你是项目主管，你如何控制风险？这种问题往往没有标准答案。项目经验部分在本书的第四部分测试实战篇的第6章，测试项目实战经验，以实战为例做了分析讲解。

3．英文

电话面试主要考查应聘者英文的听力和口语。笔试题是用英文写成，主要考查阅读能力和专业词汇量。面试时用英语交谈，主要考查听力和口语。

4．智力问答

这种题目考查的是应聘者的思维水平。例如北京市有多少辆车？有多少个井盖？还有些类似脑筋急转弯的题。这种题目未必有标准答案。回答这种题目，需要答题者说出自己的基本思路，能自圆其说、思路富有逻辑性，就达到了目的。

5．性格

对性格的考查通常出现在电话面试和面试中。有时也会在笔试中出现。答这种题时，注意要从理智出发而不是从情感出发。我个人认为这种题的意义不是很大，每个人都会伪装自己，不太可能都说出心里话。

6．个人情况

在电话面试和面试中，个人经历、简况及家庭背景等也会被问到。例如为何要辞职来新公司？对薪水有什么要求？

在以上6种类型中，1、2、3最重要，其中又以1和2最为重要。理论基础和工作实践水平，内行人迅速就可以判断出来，唬弄过关的可能性基本不存在。

2.3.2 题型

根据作者们多次跳槽过程中在多家IT外企的面试经验，笔试题型通常是选择题为主。如果

Offer

是机考，则基本不可能有选择题以外的题型。

很少有填空题。程序题中，有时程序中会空出几行，要求填入正确的语句。

阅读或写代码是肯定会出现的题型。写代码时，可能是实现一个算法，可能是实现一个函数或方法，可能是实现一个类。

小技巧

如果应聘者写不全这段代码，即使是用流程图或文字的形式，也要把主体思想标明。

非计算机化的笔试，即做卷子的情况，通常会有一两道题给出了软件界面，要求应聘者写出测试用例。除此之外，写出测试用例的题型在面试中是一定会有的。除非你应聘的是组长及以上岗位。

面试时主要是一问一答的交谈。

通常在面试的屋子里有一块黑板等能书写文字的地方，面试中通常会要求你写一些东西（当然是用英文）。例如。

- 画出项目架构。
- 给出一个软件界面，要求应聘者写出测试用例。

Offer!



第二部分 测试基础篇

在第一部分测试生存篇中，通过第 1 章对软件测试业内 20 个方面的介绍以及第 2 章对简历与面试的分析，相信对大家已经对软件测试业有了一定的认识。

从本篇开始，我们正式进入了软件测试专业领域的讲解。

我们将软件测试理论分为两大部分。

- 软件测试的基础概念与原理。
- 涉及软件测试相关的一些计算机应用领域的知识。

我们在讲解的同时，给出了一些面试题，通过细致的解析，帮助大家更好、更系统地掌握软件测试基础概念与原理。

例子

例如测试一个 B/S 结构的软件，该软件使用 Java 语言在 Linux 平台上开发，采用了 Oracle 数据库，使用了 APACHE 和 TOMCAT 服务器。

那么，作为一个测试工程师，我们必须具有 Javs、Linux、数据库、网络方面的知识，才有可能设计并进行好这个软件的测试。仅仅知道什么是集成测试，什么是回归测试，也许也从朋友处抄到了一些黑盒设计用例场景模型，是不可能做好这个软件的测试的。

因此，我们将计算机应用领域与测试基础概念原理并列，作为单独的一章。实际上，计算机应用是面试中必然会出现的考查部分，其重要性丝毫不弱于测试基础概念原理。

强调

事实证明，决定一位测试工程师薪水的高低，关键在于他的计算机应用领域水平的高低。

显然，若计算机专业科班出身，投身软件测试领域，则具有一定优势；若非科班背景，想补足劣势，则必须付出更多的努力。

本书有一个重要论点贯穿始终，即测试与开发密不可分，开发能力是测试工作的重要组成部分。请读者在平时的工作与学习中用这个思想来指导自己，保证正确的前进方向。

下面，请大家跟随我们进入测试理论的世界。

第3章 软件测试基础 1——概念与原理

3.1 软件测试的背景与概述

本节我们主要讲述软件测试工程师应当具备的素质和技术水平，然后讲述软件质量与缺陷之间的关系，接着再介绍在软件生产过程中测试人员创造了什么样的价值，最后将举例说明在软件开发模式-V 模式中软件测试活动是如何与其他开发活动进行对应的。

3.1.1 测试需要什么样素质的测试工程师

作为一名软件测试工程师需要具备以下素质。

- 适应新环境的能力：软件测试人员与软件开发人员很大的不同就在于，软件测试人员在大多数情况下都会在不同技术背景的项目中穿梭，而软件开发人员则一般都会长期地在某一种技术或具有相似技术背景的项目上进行开发。这就注定了软件测试人员在新的项目、新的技术环境中需要有很强的适应能力，且有兴趣在新的测试环境中探索新软件的功能。
- 沟通能力：沟通能力通常表现为与需求、设计、开发相关的工作人员进行技术交流的能力，发现软件缺陷后详细准确地描述缺陷细节的能力。当断定一个软件缺陷是一个重要、严重的缺陷时，软件测试人员应具备表达并坚持自己观点，说服开发人员修改缺陷的能力。
- 善于发现问题的能力：这是软件测试人员最应具备的能力之一。善于对软件的行为提出质疑、善于发现问题、仔细重现问题、确认问题是软件测试工程师入门的必经之路。
- 善于分析问题、定位缺陷：软件测试过程中经常会出现多个软件缺陷有同样的根源。而如果测试人员多次提交根源相同的缺陷时势必会加重开发人员分析软件缺陷的压力。因此，软件测试人员应当具备初步的对软件缺陷的分析能力，并能定位软件缺陷出现的范围，提高复现概率。当然，这需要测试人员不断地提高自身的业务能力和具备更多的开发技术。这也是软件测试人员逐渐走向成熟的必经之路。
- 耐性：有时软件缺陷就会出现在一些并不起眼的地方，人们甚至会觉得那个地方简单到根本不会出现缺陷，这时我们就需要有耐心的测试工程师，脚踏实地地逐个排查，不放

Offer!

过任何一个看似无价值的角落。

- 创新能力：软件测试不是证明软件可以工作，而是不断找出新的办法使软件的缺陷一一现形。
- 沉着稳重：软件测试人员不可人云亦云，要有自己的分析。因为软件测试人员应当站在客户的立场上思考问题，所以当软件测试人员对业务不熟时应当仔细询问开发人员，并不断分析，然后以自己的判断来执行测试。
- 从用户的角度看问题：在所有阶段的测试中都应当“想用户所想”，因为软件测试的目的就是验证软件的实现与需求文档一致并且确认软件产品满足客户的需求。因此，我们在遇上疑似缺陷时就可以通过多问自己几个来自于用户角度的问题来确定是否是缺陷及严重程度如何等。
- 总结经验的能力。

除此之外，作为软件测试人员还应不断提高自身技术与业务水平。

- 规划测试能力（分配人力资源、设备资源、安排测试时间）。
- 熟练运用各种黑盒、白盒测试方法设计测试用例。
- 熟练使用测试用例管理工具。
- 熟练使用缺陷管理、跟踪工具。
- 熟悉配置管理工具。
- 熟练使用黑盒、白盒自动化测试工具。
- 了解黑盒、白盒自动化测试工具原理。
- 熟悉测试阶段、测试流程。
- 熟悉主流的操作系统（和手机操作系统）。
- 熟悉主流的数据库。
- 熟悉甚至熟练掌握一到两门高级编程语言（C++、C#、Java等）。
- 熟练一到两门脚本语言（VBScript、JavaScript、Perl、Python等）。
- 熟练网络知识。
- 精通所测的软件产品的业务逻辑。
- 良好的中英文沟通能力。

3.1.2 软件质量与软件缺陷

对软件产品的高质量需求一直是软件研发组织不懈追求的目标。这样的需求通常包括以下两个方面。

一方面，软件产品需要满足客户的质量需求，如产品的功能性、有效性、可靠性、安全性、优性能等。除此之外还必须兼顾产品质量的社会效应，如避免触及宗教、道德、习俗等方面的问题。另一方面，软件产品需要满足软件研发组织的内部质量需求，如软件质量要符合软件研发组织的内部软件质量保证体系中的各项规定。除了一味地追求高质量以外，还必须结合企业

Offer!

的自身情况，在保证产品质量的同时兼顾企业的利益，缩短项目的开发周期，降低成本。如果在软件产品即将上市之前，测试人员发现了一个很深层次的软件缺陷，缺陷的重新概率较低，重现步骤较复杂，客户遇上该缺陷的可能性很小。即使遇上对企业带来的影响很小，但是修复此缺陷需要进行大量的代码改动，并有可能带来新的软件缺陷。这时，项目的负责人可以对此缺陷进行正式的评估。如果风险适当，可以放弃对此缺陷的修改，让软件产品按期上市，然后在产品维护阶段再对缺陷进行修复。

对应于软件产品的质量需求，我们可以把违背了客户的需求软件问题或潜在的可能给社会带来负面影响的软件问题看作是软件的缺陷，另外还可以把与软件的需求、设计文档不一致的软件问题看做是软件的缺陷。

所以我们可以把软件缺陷的表现方式总结如下。

- 软件的功能没有实现。
- 软件的性能指标、安全指标、可靠性指标等非功能指标不达标。
- 产品的测试结果与设计不符。
- 产品运行时出现内存泄露、应用软件崩溃及系统崩溃等情况。
- 产品的易用性低，操作复杂，逻辑混乱，客户不易掌握。
- 软件的兼容性差。
- 软件产生的数据精度不够及不一致等。

3.1.3 我们创造了什么

我们创造了价值。

软件测试工程师的目标就是尽早地找出软件中的缺陷。

软件产品这样一个看似复制/粘贴就可以生成一个新的副本的东西到底需要多少投入？

一个软件产品从无到有，再到上市需要经过以下过程。

- 收集客户需求，分析需求。
- 设计系统架构。
- 详细设计。
- 编码。
- 单元测试。
- 功能测试。
- 系统测试。
- 验收测试。
- 产品上市。

在上述过程中，需求、项目管理、设计、开发、测试人员还必须不断地对各个阶段的各种文档进行评审。在各测试阶段测试和开发人还必须重复一轮又一轮地发现缺陷、修改缺陷、发现缺陷……的循环，因此通常完成一个软件产品需要投入巨大的人力物力。例如一款智能



手机，其软件部分从需求分析、设计等一直到上市需要数百名工程师花11～18个月的时间才能完成。

对软件的测试起始于设计阶段。在该阶段测试人员就应当对设计文档进行测试，以期尽早地发现设计中的缺陷。这样的测试被称为静态测试。它的测试/修复的成本也是最低的。因为在这个阶段发现的设计缺陷只需要修改设计文档，并不会引起更多的开发人员的代码修改，此时开发人员还没有开始编码。

我们还可以试想如果一个在单元测试阶段应当被发现的软件缺陷被遗漏到系统测试阶段甚至用户验收测试阶段。那么随着软件产品复杂性的提高定位这个缺陷就会变得越来越困难。随着项目的向前推进越来越多的角色会被卷入到对缺陷的讨论，这无疑加大了沟通的成本，而且这样一个软件缺陷从被发现到被更正的周期也会被大大延长，修复的时间成本也被加大。

通常我们认为如果一个软件缺陷在设计阶段被发现只花费1元钱，随着开发测试阶段的推移，同样一个缺陷如果被遗漏到上市之后被客户发现，那么修复这个软件缺陷的费用可能会有数百或上千倍的增长，因为产品制造商轻则需要给已售出的产品打补丁，重则需要召回，甚至赔款。

所以作为软件测试工程师，应要尽早地发现软件中的缺陷，这就是其价值所在。尽早地发现尽可能多的缺陷，将会帮助项目省钱。

3.1.4 开发模式中测试阶段

测试和开发是软件生命周期中最重要的两项活动。两者互相依赖，相互促进。

软件测试阶段和软件开发阶段是同时开始的，并且同时结束。

软件项目的前期以软件的开发为主，测试为辅。软件集成之后以软件的测试为主，开发为辅。两者相互依赖。

软件的开发模式多种多样，有大爆炸模型、编写边改模型、瀑布模型、V模型（图3-1）、W模型和螺旋模型等。

以V模型为例。V模型经历需求分析、系统概要设计、（接口设计）、详细设计、编码阶段、单元测试、（功能测试）、集成测试、系统测试和用户验收测试等阶段。

在这个开发模式中设计、开发和测试之间的对应关系如下。

- 需求分析←→用户验收测试。
- 系统概要设计←→系统测试。
- （接口设计）、详细设计←→（功能测试）、集成测试。
- 编码阶段←→单元测试。

在软件项目早期的设计和开发阶段，其对应的测试团队就应该与设计、开发人员一起分析、阅读、审查对应的文档或代码，并开始着手准备测试计划、策略，开发测试用例，搭建测试环境，以及编写测试脚本等工作。而后期，测试团队的工作主要是测试、寻找软件缺陷，对前期的设计和开发文档进行验证。而开发人员的主要任务是对软件缺陷进行修复等。

Offer

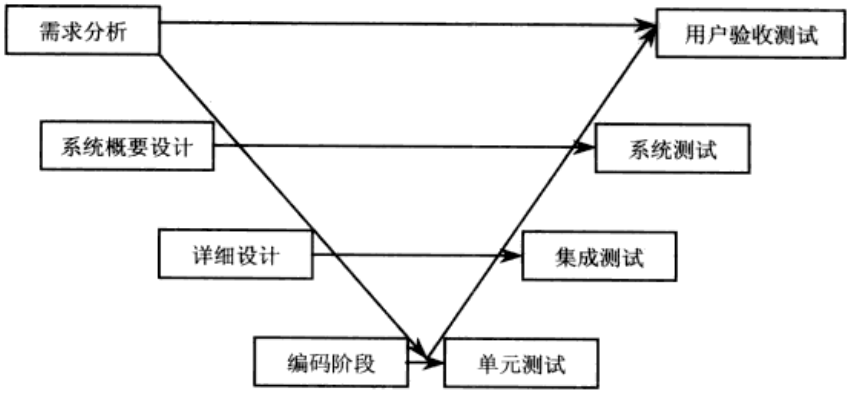


图 3-1 V 模型

3.2 测试活动的开展

本节我们主要探讨一下测试活动应该如何开展，一般会经历一个怎样的流程。一个测试活动从接手起一般经历对测试对象进行分析做出测试计划，开发测试用例，对软件产品进行测试，整理并发送测试报告等环节。本节的重点在于对以上的测试活动中的各个环节进行讲解，从而为读者提供一个测试活动开展时测试人员的任务和注意事项的参考。

3.2.1 测试计划

做任何事都需要有计划，软件测试更不例外。一个好的软件测试计划是测试活动能够顺利开展、进行的保障。而软件测试活动的行进过程中难免出现各种各样的困难和问题，因此软件测试计划也会不时地修改以适应新的问题、新的日程安排等。它不是在写好之后就一成不变的，因此我们切忌在填写完测试计划后将其束之高阁。

此外测试计划并不只是按照计划模版进行填写的一个 2～3 小时的活动，而应该将测试计划看作是一个计划的过程。对测试计划中所包含的各个部分进行详细分析、积极论证，从而把分析论证的结果编写到测试计划中。这样才是真正产生测试用例的过程。

软件测试计划在 ANSI/IEEE 标准 829/1983 中被描述为以下几个方面：规定软件测试活动的目的和被测目标、测试范围、测试方法、测试所要用到的资源和测试的进度安排；然后就是要阐明对测试目标要进行哪些方面的测试、需要执行的测试任务、每个任务的负责人；最后就是列举项目中的风险和防范风险的措施。

- 测试活动的目的和被测目标

在测试的目的中我们通常会阐明被测软件需要实现什么样的功能，在功能测试中我们将达到多少的覆盖率。在其他各项非性能测试中，软件将要达到怎样的指标等。

被测目标是最显而易见的，但也是最容易被忽视的。例如现在要测 Outlook，那么 Outlook

是什么？Outlook是编辑和接发邮件的。那还有呢？还可以订会议，还可以保留通讯人的记录为VCard形式等。对于这些功能，编写测试计划文档的人都很清楚。但是测试人员清楚吗？测试用例开发人员清楚吗？或者他们有什么不同的见解呢？开发人员和软件设计人员对此又有什么不同见解呢？

- 测试范围

在测试计划文档中应当指出我们对软件产品的哪些部分进行测试或哪些部分不进行测试。例如，被测软件产品是前一产品的升级。该产品之前的部分已经经过了详细的测试，那么这时我们就应该标明哪些部分我们是要测的，而哪些部分是不需要测的。

当然，还有其他情况也可能需要指明某些部分是不在现有的测试范围之内。例如，在产品详细说明中标明了被测软件会对一些网络异常进行处理，但是在黑盒的条件下测试人员无法完全模拟所有的网络异常。因此我们可以在测试计划文档中指出这些异常处理不在我们的测试范围之内，或指明这部分的测试由单元测试保证。

测试范围还应指明该测试所处的测试阶段。我们可以按照该软件的开发模式来对应测试的阶段，并且通常情况下，软件的测试计划是与软件的测试阶段一一对应的。然后我们需要对该测试阶段的进入条件和退出条件进行详细的描述，而不是按照日程表来安排一个产品的测试是否进入了某一个阶段，这样才能避免因测试阶段划分不清楚带来的责任纠纷。

此外，测试范围还应指明该测试包含的功能测试或非功能测试的类型，例如有的被测软件非常强调性能。这时性能测试这一块就很可能单独拿出来做一个测试计划文档，因此在该文档的测试范围中就应指出此测试活动针对性能测试。当然，也可以在一个测试计划文档中指明此测试活动包括功能测试和多项非功能测试。

- 测试方法

在测试计划文档中应指出该测试活动的所使用的测试方法是白盒、黑盒或是灰盒，或是多者兼有，并且指出测试方法的名称和在哪些情况下使用该方法。例如我们指明对文本框的输入使用等价类划分和边界值法，对手机播放软件在不同信号环境中的表现使用状态变更法等。

- 测试所要用到的资源

测试所要用到的资源包括人力资源和设备资源。

人力资源包括在测试活动中测试小组可能接触的人和测试小组中直接参与测试活动的人。我们可以将以上人员列成一张表，表中包括所有人员的姓名、职务、联系方式和职责范围等，也可以将一些职能部门或组织加入到这个表中。这样在测试活动中测试人员遇上问题时，我们就可以很容易地知道我们应该求助什么人。

对于测试小组中直接参与测试活动的人，我们可以结合测试的进度表对每个测试人员进行测试任务的安排，而且应当做到责任到人。

对于设备资源的管理应当包括对文档的归档、软件工具的归档和其他硬件设备的分配。我们同样可以将所要用到的文档和软件工具制成一张表，在表中包括文档或软件工具的名称、版本、用途和下载地址等。

Offer

其他硬件设备包括的范围很大，大到测试的场地、办公室，小到订书机、电话等，我们都需要积极地做准备，并且维持一张硬件设备使用、维护、归属情况的列表，在这些硬件设备中通常有些设备是紧缺资源，可能由多个测试小组共享。这时一定要结合测试活动的日程安排对这些设备进行预订和维护。

- 测试的进度安排

当收集完以上的信息之后我们可以开始着手对测试活动的进度进行安排了。好的测试进度的安排是灵活的测试计划的重要体现。那么什么样的测试安排才是灵活的呢？我们可以想想前面提出将某一测试阶段的进入条件和退出条件作为该测试阶段的开始和结束，这比将某一特定的日期作为限制灵活很多，因为测试活动中有很多事情是测试小组的人员无法控制的。例如在开发测试用例阶段，我们需要相关的设计文档作为输入，而设计文档若不能如期完成则必定会影响到测试用力的开发，因此可以把设计文档的完成作为开发测试用例阶段的起始点，在此基础上加上时间期限。这样就可以把我们能够控制的事情和不能控制的事情分开，从而避免了风险。

测试进度安排中还有一个比较容易忽视的地方就是对测试设备进行预订和调试安装。例如有时我们到测试开始后才发现一些不起眼的设备还没有调试好，因此在测试进度安排中应该把这部分内容考虑进去，并且要责任到人。

- 测试策略

测试策略通常是配合测试阶段来做的。当测试计划编写人员对测试阶段进行了限定之后，就可以开始定义测试策略了。测试策略通常包括该测试活动中将用到何种测试手段，白盒、黑盒或是灰盒，或是多者兼有，并且描述为什么要如此使用测试手段。当确定了测试手段后就可以确定测试方法，描述各种测试方法在测试中的适用范围和为什么要使用该方法。

此外，如果要进行自动化测试，则应当分析自动化测试和手动测试的分配情况。例如利用自动化测试覆盖 BVT（Build Verification Test）测试或回归测试，利用手动测试覆盖功能测试等。当然，同样可以阐明为什么要如此安排。

再者就是测试工具的选择。例如选择何种测试用例管理工具、何种缺陷跟踪工具、何种自动化测试工具以及何种性能测试工具等，都要一一分析并阐明理由。

3.2.2 测试用例的产生

前面介绍到好的软件测试计划是测试活动的能够顺利开展、进行的保障，那么好的测试用例就是测试活动质量的保障。一个测试用例集合的覆盖率、测试方法的选择以及测试粒度和深度的选择直接影响到了测试活动的质量。

1．何时开始编写测试用例

我们都知道测试活动开展得越早越好。在早期我们就应当对需求和设计文档进行静态测试。阅读、分析文档，对文档中不清楚的部分进行询问，对文档中的错误进行更正。这实际上就是编写测试用例的预热，因为通过前面的对文档进行阅读、分析、提问、纠错的活动我

Offer

们就可以对软件产品的需求和设计有一个较深的认识，而且这个认识会建立在设计、开发、测试人员意见一致的基础上。这样，测试人员编写测试用例时就可以减少主观的臆断。所以我们认为在对需求和设计文档进行静态测试的阶段就开始了编写测试用例的准备工作，并且此时已经形成了初步的测试用例文档。一旦需求和设计文档定稿之后，测试用例编写人员就可以开始按照需求和设计文档中确定的设计对测试用例文档进行修改和补充，最终形成符合最新设计的测试用例。

之后我们就需要对需求和设计文档文档条目与测试用例之间的对应关系进行梳理。最后形成一张对应表格，如表3-1所示。

表3-1 建立需求或设计文档与测试用例的对应关系列表

	需求条目1	需求条目2	需求条目3	需求条目4	需求条目5	……
测试用例1	√		√			
测试用例2		√				
测试用例3	√			√		
测试用例4						
测试用例5			√		√	
总 覆 盖	√	√	√	√	√	……

使某一需求条目或设计条目的修改可以很快定位到其对应得测试用例，以对其进行修改。而在测试时，也可以很快地看出每条测试用例覆盖了哪些需求或设计条目。

2．选定测试方法

在编写测试用例的阶段需要对测试的方法做出选择。在选择测试方法时通常采用自顶向下的方法。

首先是对测试用例适用的测试阶段进行划分，例如单元测试、集成测试、系统测试、用户验收测试等。然后在每一个测试阶段中再对测试用例的测试目标进行划分，例如在单元测试阶段要进行白盒测试，在系统测试阶段要进行功能性测试、性能测试等，这也就形成了每一个测试用例的测试目标。再继续往下分就可以开始分析为了实现某一个测试目标应该使用何种测试方法了。例如在白盒测试中，我们要用语句测试来覆盖所有语句，用基本路径测试来覆盖所有逻辑路径，又如用等价类划分和边界值法来覆盖功能测试中界面的输入框等。

3．测试用例的内容

每一个测试用例至少需要给出以下测试信息以供测试人员能够正确的完成测试：测试目标、测试对象、测试环境设置、前提步骤、输入数据、操作步骤和期望结果。

- 测试目标：指测试用例的目的，是对测试对象进行哪方面的测试。如功能性测试、性能测试、易用性测试、压力测试等。
- 测试对象：指测试用例所测试的对象，如软件产品中的某一模块的某一功能。又如在单元测试中，测试对象可以是代码中的一个类、函数等。在性能测试中，也可以是软件产品的某一个性能指标。

Offer

- 测试环境设置：指在执行该测试用例时,需要对测试环境进行的（特殊）配置。例如在网站测试中，配置操作系统和浏览器的搭配及浏览器的配置。又如在性能测试中，配置模拟器以模拟网络中的流量、延时、带宽等情况。
- 前提步骤：指在执行该测试用例时，需要把软件事先运行到某一初始状态的步骤，然后再对软件进行针对该测试用例的测试。
- 输入数据：指在执行该测试用例时需要事先准备好的测试数据。例如在执行性能测试时需要准备的输入数据。又如在 VCard 与电话簿记录的对应转化测试中，需要事先准备大量的 VCard 文件以备在测试时倒入手机中。对于此类的输入数据及其比对结果，测试用例编写人员一般都应该事先编写好，并以附件或连接的形式附加在测试用例中。这样的耦合的形式方便了测试输入数据的升级，避免了对测试用例本身的修改。
- 操作步骤：指在执行该测试用例时，从前提步骤设置完成到比对期望结果之间需要运行的测试操作。
- 期望结果：即该测试用例的验证点。通常每个测试用例设一个验证点，验证点过多容易造成测试用例覆盖面过大，甚至出现在多个测试用例中重复的验证点，这样的结果就是一轮测试完成之后的测试报告的数据可能会不准确。例如一个测试计划集合中包含了 100 个测试用例，其中有 5 个测试用例重复地覆盖了一个验证点。这种情况下，当这个验证点被验证有缺陷时就会使 5 条测试用例无法通过测试，这明显影响了测试报告中的数据反应软件真实质量的能力。此外，测试用例的期望结果还应该注意其描述的准确性，避免产生二意。

4．测试用例的粒度

测试用例的粒度通常对测试计划的执行有很大的影响。在保证测试覆盖面的情况下，粒度越大，测试用例开发编写人员的工作量越小，测试用例执行周期越短，但对软件产品的质量保证能力越差；粒度越小，测试用例开发编写人员的工作量越大，测试用例执行周期越长，但对软件产品的质量保证能力越强。

如果测试用例的力度非常粗，有时非常细节的设计条目有所修改也不会引起到这类粗粒度的测试用例的修改（即使该用例是用来覆盖被修改的设计条目）。但是这类测试用例在被执行阶段给执行者很大的自由空间，以至于不同的执行者因其理解不同在同样的产品上执行多次会获得多个不同的结果。这对于测试新手来说一般不是件好事。

如果测试用例的力度非常细，则所有其对应的设计条目的修改都会引起到这类细粒度的测试用例的修改，这无疑加大了工作量。当然，这类测试用例规定了测试人员的执行细节，即使是测试新手也能按部就班地执行这类测试用例。

因此我们应当设计粒度适当的测试用例，这样就可“进可攻、退可守”。

5．测试用例的组织方式

测试用例的编写一般都采用测试用例套件的编写和组织形式，这是一种非常自然的组织形式，正如我们之前在“选定测试方法”小节中介绍的。我们会自顶向下将测试用例集合分为适

Offer

用于某一测试阶段的集合，然后将某一测试阶段的测试用例集合按照不同的功能进行划分，之后再将针对某个功能的测试用例按照组件的方式进行划分。组件之下就是测试套件了，一个组件可以有一到多个测试套件，如图3-2所示。

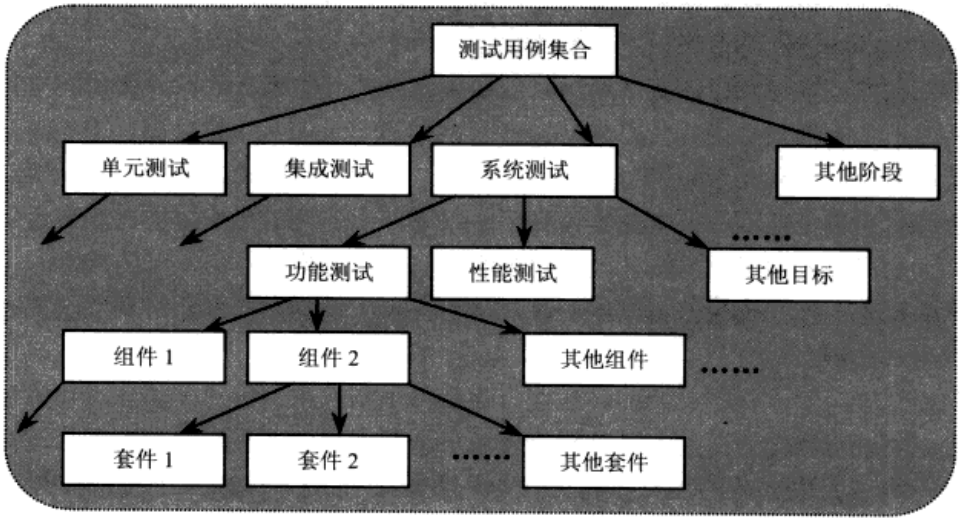


图3-2 测试用例的组织方式

6. 测试用例的审查

与其他的文档审查一样，测试用例的审查也是相关人员意见达成一致的过程。当测试用例开发完成以后，用例开发人员就应当开始着手组织开发和测试人员对测试用例进行正式审查。

正式审查会议的组织人员需要在事先订好会议室，并至少提前一周将文档通过邮件等正式的形式发送给参与审查的角色，以便各角色对文档进行阅读，发现错误，并在文档发出到审查会议开始的这段时间内定期地提醒参与审查者提交发现的问题和文档中写得不够清楚的地方。在收集完问题之后，组织人员应当积极地对这些问题做准备。

在审查会议上要对前期收集的问题一一作出答复。遇到会议上提出的新问题应当详细解答，如果会议上不能立即解决的问题应当将其记录下来。在审查会议结束之后要将会议中解决了的问题、达成的一致的问题和未解决的问题进行归档，并安排责任人对其进行跟踪，然后将以上归档和责任人的安排制成会议记录，再将会议的记录发送给与会者和其他相关人员等。

待所有审查中发现的缺陷都得到更正、所有问题都被解决以及各方的意见都达成一致之后，测试用例开发人员就可以将通过审查的测试用例文档定稿了。

7. 测试用例的有效性验证

当新的测试用例文档定稿之后，我们还有必要对测试用例进行一次有效性验证。这个过程可以在软件的非正式测试版本上进行。由于测试结果并不作为任何正式测试轮次的报告，其作用只是验证测试用例与待测产品的一致性及测试用例中所描述的测试环境等的有效性，所以测试用例的有效性验证中并不要求将所有测试用例进行测试，只需要有代表性的挑选几个能体现产品特性和环境有效性的用例进行测试即可。

Offer!

3.2.3 开始执行测试

经过以上几个小节的准备工作，测试人员就即将进入软件测试用例的执行阶段了。这时执行测试用例活动的组织者（通常是测试组长）就应该开始安排测试轮次用例集合和测试用例执行进度，分配测试任务。

1. 挑选测试用例组成测试轮次用例集合

软件测试用例的执行有时不是一蹴而就，而可能会分为好几个轮次。测试轮次用例集合指的是该轮次中需要执行的测试用例的集合。各轮次的测试用例集合间可能使用的是同样的测试用例，也可能是完全不同的测试用例。上一小节中我们完成的所有的测试用例套件在实际使用时通常作为一个测试用例的仓库。测试轮次用例集合中的测试用例可以是从测试用例仓库中挑选出来的一个或多个完整的测试用例套件的组合，也可以是从一些测试用例套件中挑选出来的测试用例的集合。例如，在为软件的某一组件进行首次功能测试时，可以将这个组件的所有功能测试用例套件中的测试用例作为一个测试轮次的用例集合使用。又如，在为一个组件进行软件功能的回归测试时，因为回归测试主要是针对软件的改动而进行的，所以我们可能会从这个组件的所有功能测试用例套件中挑选一些与本次改动有直接或间接关系的测试用例（详细内容可参考回归测试一节）作为一个测试轮次的用例集合使用，而非所有的功能测试用例套件中的测试用例。

这样的测试用例的挑选是基于测试阶段、测试目的、测试日程安排、人力和物力资源情况、测试用例的覆盖面等诸多资源来考虑的，这也是测试计划和测试策略的一部分。因此，可以看出测试计划和测试策略不是一旦编写完成之后就束之高阁，一成不变的，因为在测试活动的进行中或多或少会遇上一些突发的情况，这样就不可避免地修改测试日程安排、人力和物力资源分配等、从而需要修改测试计划和测试策略。

测试轮次用例集合准备好之后执行测试用例活动的组织者就可以把集合中的测试用例分配给参与执行测试的测试人员了。

2. 正式执行测试前的软件版本验证工作

- 开发团队的版本验证工作

当开发人员的阶段性代码编写工作完成之后，开发人员需要将新开发的代码和现存的代码集成并进行一遍完整的单元测试。当单元测试通过之后，开发人员需要将所有的代码进行编译以生成可执行的软件镜像，需要注意的是，这时的可执行软件镜像并不是可正式发布的版本。开发人员还应该对其进行冒烟测试，其原则是挑选该产品最具代表性的测试用例和新改动部分的代表性的测试用例作为冒烟测试的测试用例计划。忽视冒烟测试的风险在于，当未经过任何测试的可执行软件镜像被作为新的软件版本直接交付于测试团队时，如果新的版本出现大面积退化，甚至是重大产品功能不能使用，则这个软件版本将会被测试部门直接退回。这样会增加大量增加开发团队和测试团队的沟通成本和时间成本，而且有损开发团队的声誉。

- 测试团队的版本验证工作



在新的可执行软件镜像通过冒烟测试之后，该镜像就成为测试团队的可测版本的一个候选版本。所谓可测版本就是指该软件版本没有重大产品功能质量问题并且新开发的产品功能无重大质量问题（如新功能无法启动或极度不稳定等）等。测试团队需要将一个刚接手的可执行软件镜像作为可测版本的一个候选版本的原因在于，测试团队并不能保证新的可执行软件镜像一定符合本团队的可测版本的要求且适合正式的测试环境。因此，此时测试团队会将这个可执行软件镜像列为可测版本的一个候选版本，并对其进行版本验证测试（BVT-Build Verification Test）或称作 Sanity Test。其测试目的就是验证候选版本是否符合可测版本的上述要求。如果不符合，则候选版本将被退回开发团队。如果通过了版本验证测试，则该可执行软件镜像就成为了测试团队的可测版本。

例子

不做版本验证测试是很容易带来惨痛教训的。笔者曾经经历过一个项目，该项目一开始没有任何的标准测试流程，当然也就没有版本验证这一环节。当时开发人员承诺每天出一个版本给测试团队测试，于是当测试人员拿到第一个测试版本时就开始在多台服务器上进行安装。但未曾想到的是所有的机器都无法成功安装新的版本，并且当安装包执行完毕后服务器就开始不断重启，连安全模式也无法进入。正因为如此，第一个测试版本被退回了开发团队，而且留给测试团队一大堆重装服务器以及重建测试环境的工作。这些重复的工作本来是可以避免的。

有了这个教训之后，开发和测试人员就开始在每一个新的版本正式测试之前进行软件版本验证工作了。这样的版本验证测试只会在少量的服务器上进行，并执行很少的一些为版本验证测试专门挑选的测试用例。因此大大地避免了由版本失败带来的大量的人力和时间的浪费。此后开发人员的出版本的速度也降到了一周一个，因为他们在做版本验证测试时通常会发现一些自己也认为不可轻易放过的缺陷。此后的工作就变得对测试人员非常有利了。

3. 何时开始执行测试

当新的软件版本通过了开发团队和测试团队的版本验证后，测试人员就可以开始按照该测试轮次中所分配的测试用例对软件产品进行测试了。

3.2.4 报告测试缺陷

执行测试用例阶段也就是测试人员发现软件缺陷的重要阶段。一旦测试人员确定一个软件的问题是软件缺陷时，测试人员就有义务将其归档，并对该软件缺陷进行跟踪。大多数软件生产企业都会有一个网络化的缺陷管理跟踪系统，企业内部的各种缺陷都可以存档于这个系统中。这些缺陷并不仅限于软件缺陷，还可以是需求和设计文档的缺陷，同时也可以是硬件缺陷，例如制造嵌入式产品的企业；甚至可以是对现有工作流程所提的缺陷。

1. 将软件缺陷进行归档并对其进行跟踪的意义

- 缩短软件缺陷被修复的周期

Offer

开发人员必须通过被归档的软件缺陷的描述对缺陷进行复现、分析和修复。简练、准确、信息完整的软件缺陷描述可以大大缩短软件缺陷被修复的周期。

- 反映软件的质量

每一轮次的测试完成之后，测试人员都会生成该轮次的缺陷列表和该轮次测试用例的通过情况列表，根据这些列表所反映的情况可以对软件的质量情况做出客观的评估。因此，软件测试人员有时也被称为软件项目的探路明灯。

- 经验总结

无论是测试完成后所生成的表格中的数据还是每一个被归档的测试用例，都可以被作为以后新的工作的基石。

2. 既然软件缺陷归档如此的重要，那么当软件测试人员发现了一个软件缺陷时应该如何有效地对缺陷进行报告呢？

- 在报告一个软件缺陷时应该尽量避免重复报告

有些软件公司提供了强大的搜索工具来方便测试人员查询是否已存在与自己即将报告的缺陷重复的记录。这是一个非常好的实践，因为大量重复的报告同一个缺陷不仅会增加开发人员阅读和分析缺陷报告记录的时间，还会大量增加开发人员和测试人员之间无谓的沟通次数。

- 简练准确的在缺陷的标题中描述该缺陷

很多公司的缺陷归档和跟踪系统对缺陷的标题都有字数的限制，其目的在于养成测试人员能够以简练的语句描述缺陷的特征。这样，开发人员在极短的时间内通过查看缺陷的标题就可以判断该缺陷是否已经有人报告过，如果没有则能很快定位软件缺陷产生的位置和现象。

- 一个缺陷报告记录一个缺陷

这样有利于开发人员的工作开展。如果一个缺陷报告记录了两个缺陷，而两个缺陷又属于不同的开发小组，这时通常会造成开发人员的修复工作无法开展，因为很难给这样的缺陷定一个唯一的责任人。

- 信息完整

测试人员应该将测试用例所用到的测试数据和测试执行过程中所产生的一些文件作为附件附加在缺陷报告中。

- 直观形象的描述缺陷，提高缺陷重现率

详细、精确地提供重现步骤并附加一些图片、视频等信息有助于开发人员更直观地了解相关缺陷。

- 客观描述

报告软件缺陷时应当客观描述现象，避免主观的评价，更不应该对开发人员的技术水平等做出评价。

3. 缺陷报告的内容

当测试人员发现了一个软件缺陷时，就需要开始着手把这个软件缺陷录入到缺陷管理跟踪系统中。录入一个缺陷报告一般需要填写下列内容，如表 3-2 所示。



表 3-2 缺陷报告内容

缺陷属性	含义
缺陷唯一标识	缺陷管理跟踪系统中对于每个缺陷的唯一标识，通常为系统自动生成
报告时间	缺陷报告生成的时间，通常为系统自动生成
状态	标识一个缺陷当前所处的状态，是缺陷生命周期中的一环。通常随着缺陷的生命周期的演化而变更
缺陷报告人	填写缺陷报告的测试工程师
标题	缺陷的总结，要求简练准确。让开发人员在极短的时间内通过查看缺陷的标题就可以判断该缺陷是否已经有人报告过；如果没有，则能很快定位软件缺陷产生的位置和现象
所属项目	被测产品的项目名称
产品	被测产品的产品内部名称
组件	缺陷所在的组件
版本	缺陷所在的软件版本
严重度	客观地从客户的角度描述缺陷的严重程度。一般分为 5 级，通常由测试人员确定
优先级	缺陷被修复的优先级别。一般分为 3～5 级，通常由项目管理人员、开发组长、测试组长共同确定
发现阶段	缺陷被发现时的测试阶段
起源阶段	该缺陷本该在何测试阶段被发现。例如，一个继承测试阶段应该发现的组件接口方面的缺陷却被遗留到了系统测试阶段才被发现
当前负责人	缺陷在生命周期中的某一个阶段的负责人。随着缺陷状态的改变，缺陷的负责人会从测试人员（如 Open 状态）变成项目组成员（如 Assigned 状态），然后变成开发人员（如 Analysis 或 Fixing 状态），最后又回到测试人员手中（如 Verified 状态）等
详细描述	直观、形象、完整地描述缺陷，提高缺陷重现率。详细精确地提供重现步骤，附加一些图片、视频等信息有助于开发人员更直观地了解相关缺陷

在上述需要填写的内容中，详细描述一项的内容是开发人员是否能从缺陷报告中得到准确复现信息的关键。因此，详细描述一项的内容应该做到条理清晰、描述精准简练、突出重点。

在有条件的情况下缺陷报告中可以附加图片或视频（若不能拷屏，随身的手机是一个好帮手）以使开发人员能够在最短的时间里重现缺陷。

标明缺陷的复现概率可使开发人员在重现缺陷时更好地掌握尝试次数。

将发现缺陷人员的最有效联系方式写在显著的位置以便开发人员在需要时能够第一时间与发现缺陷人员沟通。

同一项目的测试人员所报告的缺陷的详细描述一项应该有统一的格式。以下我们列举了一种格式以供参考：

```
[缺陷简述（Summary）]
……
[缺陷描述（Description）]
……
[初始条件（Initial Condition）]
……
```

Offer

[重现步骤（Steps）]
1.……
2.……
……
[期望结果（Expected Results）]
……
[实际结果（Actual Results）]
……
[发生频率（Occurrence Rate）]
百分比（Percentage）或重现次数/尝试次数比（Occurrence/Attempt）
[附加文件（Attachment）]
……
[缺陷汇报人联系信息（Contact Information）]
姓名（Name ）
电话（Telephone No.）
邮件地址（Email ID）
测试组（Team）

4．缺陷的跟踪

当测试人员完成了一个缺陷的报告工作时，即表明了这个缺陷的生命周期开始了，此时项目组人员就有义务对这个缺陷进行跟踪。图 3-3 中列举的是在某一缺陷管理跟踪系统中一个缺陷从被报告到被关闭或被终结的所有生命状态及状态的可能变迁路径。

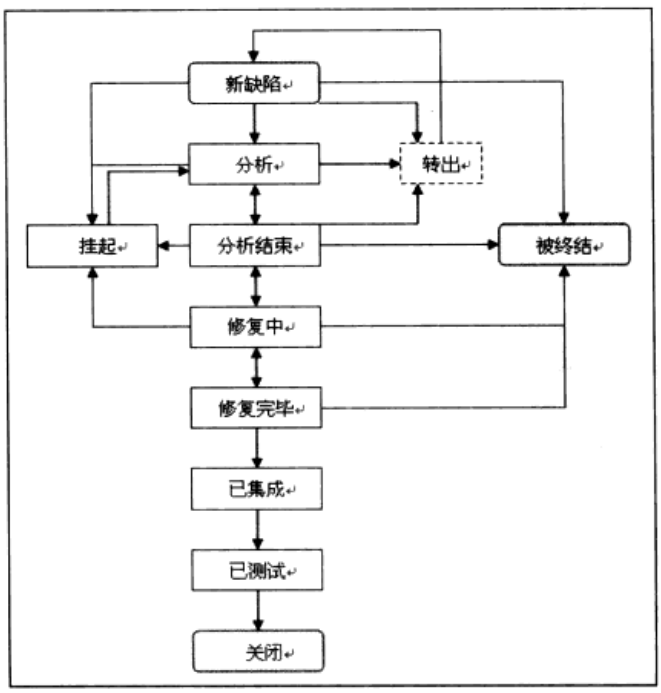


图 3-3 某一缺陷管理跟踪系统中一个缺陷从被报告到被关闭或被终结的所有生命状态及状态的可能变迁路径

在图 3-3 中，有些状态是可以互相转换的，如分析与分析结束、分析结束与修复中、修复中与修复完毕。

挂起状态通常是由于缺陷在某些状态中因缺少信息或不能重现等原因导致缺陷记录无法被继续正常处理而被转为挂起状态。当以上条件得到满足，开发人员得到了更多的信息或获得了重现的方法时，缺陷就可以从挂起转入分析状态以供开发人员开始分析。

一个缺陷有可能因为最终测试和开发人员都无法复现或信息不够或与其他缺陷重复等原因被置为被终结状态。

如果一个开发人员或项目组成员在分析缺陷的过程中发现这个缺陷是属于其他不属于其负责范围的组件或是属于其他项目组，该缺陷就会被转出，其状态也会被置为新缺陷，注意，此时转出是一个动作而不是一个状态。

3.3 黑盒测试及测试用例设计方法

黑盒测试这个看似简单的问题，在面试中被提及的机会真是数不胜数。特别是应聘者只有一两年工作经验时，面试官通常会问道。

- 是否做过测试？
- 黑盒还是白盒？
- 怎么做？
- 有什么区别？

这么简单的问题可能大家早早就预备好答案了。还需要一遍又一遍的讨论吗？其实越看似简单的问题，如果应试人员能提供深刻的分析和独特的见解，则越能给面试官留下深刻的印象。我们需要时时刻刻提醒面试官“我不光是懂，我是真懂”；我做黑盒测试不是在那里点点鼠标键而已，我是有目的地点鼠标键。

下面让我们看看我们为什么是有目的地点鼠标吧。

测试和开发就像软件项目中的两条腿，时而交替时而并行地推动着项目前进。写代码是开发人员的主要任务之一，相比之下测试用例的开发就是测试人员的主要任务之一。所产生的测试用例的质量也是考核测试用例开发人员业务水平的重要标准。在写作用例方面不断进行质的提高，是一名普通测试工程师成为高级测试工程师的必经之路。

软件测试通常分为黑盒测试和白盒测试两类。本节我们将重点介绍黑盒测试及其测试用例的设计方法。

3.3.1 黑盒测试定义

面试试题 1

什么是黑盒测试？



解答

黑盒测试就是测试人员把软件产品（或阶段性产品）看做是一个黑盒。在测试过程中测试人员只需关心对这个软件黑盒进行操作会得到什么样的结果，而不必深入地去了解它的内部实现机制所进行的测试活动。这样的操作可以是字符串的输入、图片的输入、视频流的输入，甚至可以是事件输入。而不同的输入进入相应的处理软件（即这个黑盒）进行处理之后通常能得到不同的结果，如图 3-4 所示。

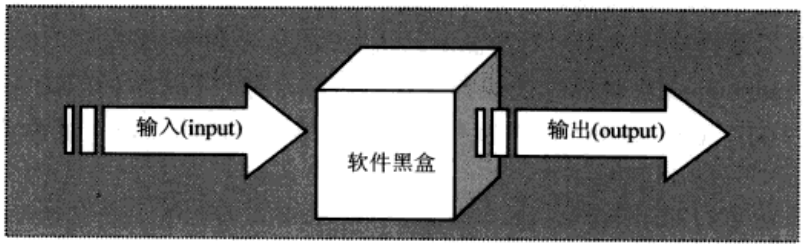


图 3-4 黑盒测试

例子

例如我们在 Windows 的命令行中输入字符串 dir 就可以得到当前目录下的子目录及文件的列表。而输入 tasklist 后就可以得到一张正在系统中运行的任务的列表。在以上的操作中我们都不必去考虑命令行解析器会如何解析输入的字符串，也不必考虑系统如何获取我们想要的信息并如何把它显示在屏幕上，这就是黑盒处理机制。我们只关心输入（input）的和想要得到的输出（output）。如果在初始条件确定的情况下的一组确定的输入经过软件产品这个黑盒进行处理后并没有得到期望的结果（expected result）时，则说明此时就发现了一个软件的缺陷（defect）。

面试试题 2

为什么要做黑盒测试？

解答

- 验证（verify）软件产品是否符合需求文档的设计（IEEE 1983 of IEEE Standard 729）
- 证实（validate）软件产品符合终端用户的需求（IEEE 1983 of IEEE Standard 729）

面试小技巧

在 IEEE 对于软件测试目的的论述中，各项软件测试活动都是围绕以上两个目的展开的。所以在面试过程中，当面试官问及软件测试的目的时不妨把以上两点作为答案，既简练，又深刻。

我们把字符串 dir 输入到其他软件中（如计算器）可能毫无意义，但在 Windows 的命令行中就能被解析为获得当前目录下的子目录及文件的列表，并且在初始条件确定的情况下将得到可预测的输出。这样的输出在软件设计阶段通常就被定义下来以保证开发人员编写的程序有章

Offer

可循。这正是软件测试的目的之一：**验证（verify）软件产品是否符合需求文档的设计（可参考IEEE 标准 729 之 IEEE 1983）。**

此外，在黑盒测试活动中，测试人员只按业务逻辑测试而不需要考虑内部实现。这就很好地模拟了终端用户的行为。然而终端用户的行为并不会都在软件需求文档中定义。例如测试一个网站，我们可能会模拟黑客对网站进行一些攻击，但是黑客的攻击方式是无穷尽的，正所谓是道高一尺魔高一丈。黑客们总会想出新点子来达到他们的目的。既然攻击方式是无穷尽的，那么我们就不可能在我们的需求文档中定义应对所有攻击方式的策略。这时我们的测试就派上用场了。我们可以尽量模拟终端用户对产品网站进行攻击。这样的测试既可以是事先定义好的，也可以是随机的（adhoc test）。像这样的模拟终端用户操作对产品进行的测试活动就是在履行软件测试的另一个目的：**证实（validate）软件产品符合终端用户的需求（请参考 IEEE 标准 729 之 IEEE 1983）。**

面试试题3

你在软件生命周期中的哪些测试阶段用到过黑盒测试？

解答

- 单元测试（unit test）。
- 组件测试（component test）。
- 集成测试（integration test）。
- 系统测试（system test）。
- 用户验收测试（user acceptance test）。

通常来说，软件生命周期中的测试阶段按时间顺序可分为单元测试（unit test），组件测试（component test）、集成测试（integration test）、系统测试（system test）及用户验收测试（user acceptance test）。在各个测试阶段之间通常还会设置通过条件（pass criteria）和准入条件（entry criteria）。

不过由于产品种类不同，项目的大小也各不相同，所以各个公司都会对上述的测试阶段进行裁减或重新定义。例如做手机类的公司就通常把系统测试阶段再次分为系统测试（system test）和场地测试（field test）。这是因为对于手机这样的产品，其质量极大地受到手机产品与现有的移动运营商的网络的兼容性的影响，所以在内部系统测试的中期，手机的系统达到了一定的稳定性的情况下开始做大量的实网测试（场地测试）是必须的，也是同等重要的。

那么我们的黑盒测试适用于哪些测试阶段呢？其实一般来说适用于所有阶段。对于这句话大家可能最有异议的就是单元测试阶段。一般大家认为单元测试阶段应该采用的都是白盒测试方法，而非黑盒。其实不然，黑盒测试其实在此阶段是一种非常行之有效的方法。如果运用得当则可以减少很多把软件缺陷遗漏给其他测试阶段的机会。我们现在知道黑盒测试把软件产品看做是黑盒，测试人员只关心对其进行的输入和获得的输出，而并不关心其内部实现机制。而

Offer

在单元测试中，其实函数也可以被看做一个黑盒，一个小尺度上的黑盒。这个黑盒一般也被称作函数黑盒。一个组件或应用软件正是由一个个这样的函数黑盒组成的大黑盒。一个函数一般分为 4 个部分：函数名、形参表、函数体、返回值。函数名是这个函数的识别标志，一般要取的通俗易懂，且要与上下文一致。函数体是这个函数的实现机制，它是这个函数的核心部分，其决定了这个函数是否做了正确的事，是否占用了最少的代码空间和运行空间，是否用了最少的运行时间。可以这么说，函数体就是这个黑盒。而形参和返回值就是这个函数黑盒的输入和输出。利用黑盒测试用例设计的方法对其输入值域和输出值域进行分析可以极大地保证函数的质量。在本节此后的分析中将举例对此进行详细深入的探讨。

说明

一个函数除了输出返回值之外还可能有其他的输出：

- 函数体有可能处理了函数域以外的全局变量，这时全局变量也成为了这个函数输入和输出的一部分。
- 函数的参数中可能有引用型形参（C 中可用指针作为引用型形参，C++中可用指针或引用作为引用型形参，C#/Java 中引用类型的引用或值类型的引用作为引用型形参）。这时被引用的对象实例有可能在函数中被修改，从而成为函数输出的一部分，如图 3-5 所示。

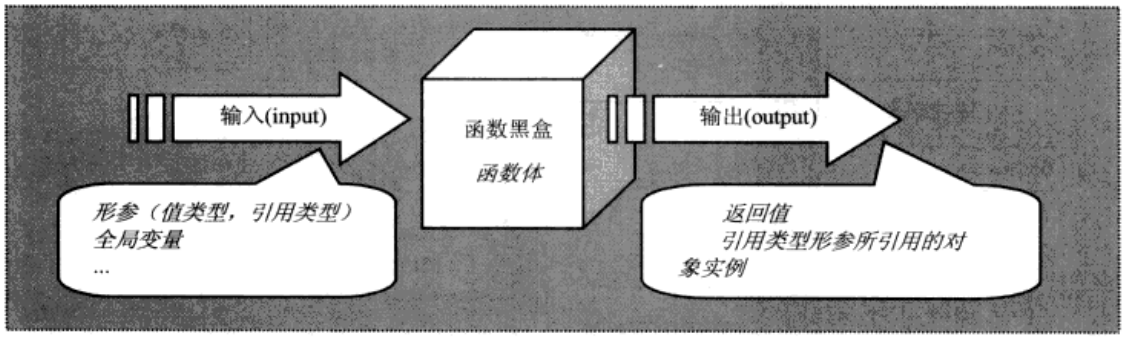


图 3-5 函数黑盒

引申

笔者认为若想做到占用最少的代码空间和运行空间，占用最少的运行时间，就应该深入了解编程语言，熟读数据结构和算法方面的书籍。专业性的软件企业如微软、Google 等就很看重这些，即便是招收测试人员也如此，有志于去这些公司的朋友有必要在此多花时间精力。关于函数是否做了正确的事，通过学习测试的知识会有不少的帮助。

正如我们前面所说的组件或程序就是通过层层调用函数来处理用户需求的，所以当多个函数能够封装成类，进而被组装成一个组件时，也就是把一个个小的函数黑盒组装成了一个大黑盒，此时就可以对它进行一定的组件测试了。这时的组件即使不能完成一个完整的功能，也能通过编写**驱动函数**或**桩函数**的方法去测试它。

Offer

面试试题4

什么叫打桩，什么叫驱动？

说明

驱动函数： 在测试被测函数时，一般需要在一个函数中构造一些实验数据作为传入被测函数的实参，然后再在该函数中调用被测函数，之后检验测试结果。这样的函数被称为驱动函数（一般为 main 函数）

桩函数： 在测试被测函数时，被测函数可能要调用一个未被实现功能的函数（或一连串函数的调用）。这时测试人员可以写一个临时的简单的函数以模拟被调函数的部分行为。这样的函数被称为桩函数，如图 3-6 所示。

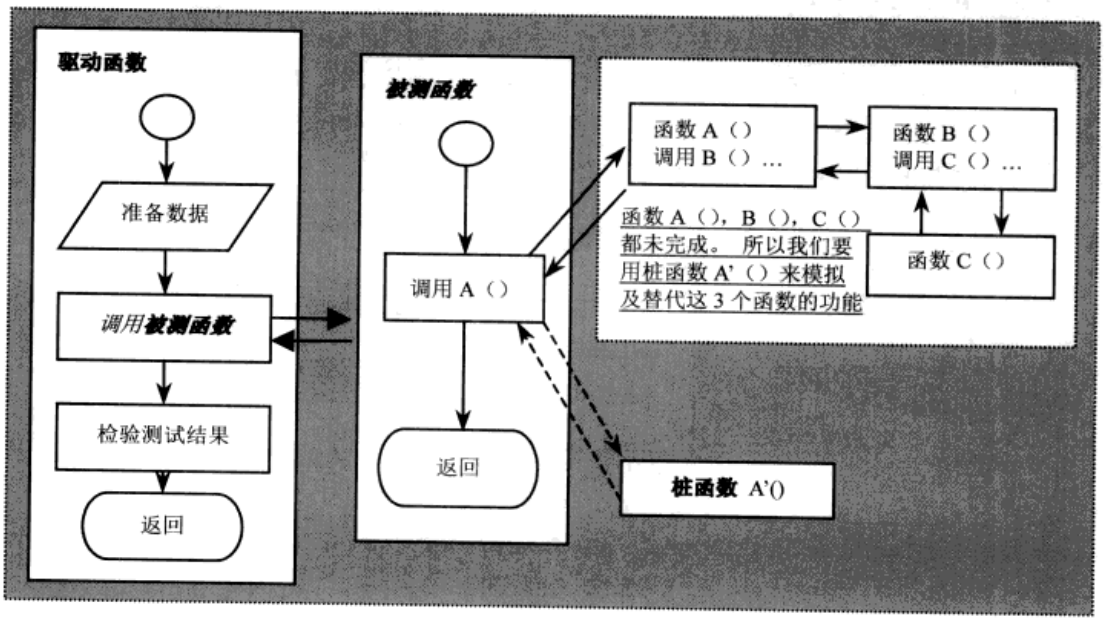


图 3-6 驱动函数与桩函数

我们通过组件暴露的公共接口分析它的输入值域，然后根据其输入值域编写测试数据。之后在驱动程序中调用组件的公共接口，并把测试数据作为参数（或全局变量）传给被测组件。[illegible]得该组测试数据经处理后所产生的输出。此后就要分析这些输出是否是期望结果以确定该组[illegible]是否有缺陷。

[illegible]后被多个测试过的组件将被集成，成为更大的组件直到成为功能齐全的应用程序。这时系统测试就将开始了。从黑盒测试的角度看，这时的软件产品就是一个更大规模的黑盒，而且因为功能齐全，我们就可以直接通过图形用户界面、命令行或该软件产品所提供的其他方式直接对对其进行黑盒测试了。

Offer

进入用户验收测试阶段，黑盒测试也是最主要的测试方法，只不过此时所做的测试大都是从用户的角度出发，而且生产企业大都会邀请用户直接加入到用户体验测试中来，以使软件产品更符合用户的习惯。

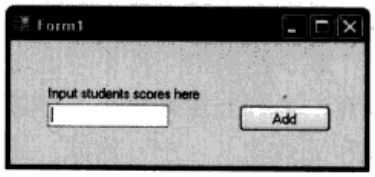
3.3.2 黑盒测试方法

1. 输入值域和输出值域

上一节我们介绍到当测试人员对软件产品进行黑盒测试时，需要对软件黑盒进行输入用以产生相应的测试结果。这样的输入值的组合就构成了我们对某个软件产品进行黑盒测试时所用的测试用例。所有输入值组合的集合就构成了**输入值域**。

我们来看一个简单的程序：

这个小程序只有一个输入框和一个按钮。我们用这个小程序模拟学校里学生们的成绩数据库录入界面。这样我们可以知道学生们的成绩一定是小于等于 100 而大于等于 0 的数。如果出现了得半分的情况，这个数的小数点后面就可以出现五，而且一般小数点后面除了五就不会有其他数的情况了（此时假设不会出现类似于 0.3 分的情况）。



那么现在我们总结一下以上的情况。这个被输入的数是一个实数，这个实数可以没有小数部分。如果有小数点后只能为 5 或零，而这个实数的取值范围是大于等于 0 且小于等于 100，则在计算机中这个数的有效范围将可以被表示为一个大于等于 0 且小于等于 100 的浮点数。这个浮点数可以没有小数部分。如果有小数点后只能为 5 或零，则符合以上条件的浮点数将被允许输入到学生们的成绩数据库中。而这样的一个有效范围就被称作**有效输入值域**。我们可以轻松得出在这个有效输入值域中有 201 种输入的组合{0，0.5，1，1.5，…，99，99.5，100}。

我们还能有其他的输入吗？当然有。为了达到测试程序健壮性的目的，可以取浮点数中的任意数值对其进行测试。我们知道现实生活中实数的取值范围是无穷大的，而且是连续的，但在我们的计算机中浮点数 f 是 32 位的。它不能像我们的生活中的实数一样，是连续的。它也只有 2^{32} 种取值的可能，因为它只有 32 个 bit 位。因此当我们在浮点数域中对这个程序进行测试时，我们将可以找出 2^{32}-201 个无效的输入组合。所有的无效的输入组合就组成了我们在浮点数域的无效输入值域。而在这个域中任何一个输入组合被输入到程序中，这个程序都必须反馈一个合理的信息以通知使用者这个输入是超出有效范围的。

还有其他输入吗？当然有。既然输入框（textbox）可以允许用户输入字符，那么我们可以满键盘地找到所有看得见的字符（非浮点型字符常量的组合），然后“爆”输一番。这样的组合有多少呢？当然是无穷无尽了。这样我们就可以看到在这种情况下我们的无效输入组合数量就扩大到无穷大了，而此时这种情况下的无效输入值域也就变得无穷大了。在这个域中任何一个输入组合被输入到程序中，这个程序都必须反馈一个合理的信息以通知使用者这个输入中含有无效的字符。

还有吗？当然还有。我们可以不输入任何值而直接按按钮。这也是一个无效的输入。而对

应于这个输入，程序就必须将输入为空的信息反馈给使用者。

综合以上3种情况，我们可以看出对于这个小程序我们的无效输入组合数为 $2^{32}-201+\infty+1=\infty$。这也就是我们的**无效输入值域**了。

结合上述分析所得的**有效输入值域**和**无效输入值域**不难看出，这两个域中的输入组合的总和涵盖了所有可能的输入组合，亦即等于整个**输入值域**，并且有效输入值域和无效输入值域无交集。所以我们有：

输入值域 = 有效输入值域 + 无效输入值域

作为对应，我们不难发现每个被测的软件组合还有一个**输出值域**，而且同样可以推理得出：

输出值域 = 有效输出值域 + 无效输出值域

通过以上的分析我们知道输入（出）值域包括有效输入（出）值域和无效输入（出）值域。每个域中都有很多输入（出）组合。在以上的例子中，我们列举了有效输入值域201个，无效输入值域无穷个。这样的情形对于我们测试工作来说可不是一件好事。

软件测试的工作不可能通过逐个枚举输入（出）值域中的输入（出）组合来完成。我们必须有一系列方法来减少工作量。首先，我们已经把输入（出）值域划分为了有效输入（出）值域和无效输入（出）值域。如果能够把有效输入（出）值域和无效输入（出）值域进一步划分为更小的集合，而且在每个集合中的输入（出）组合对于这个程序来说都是完全等效的，那么就可以在每个集合中挑选一个或几个输入（出）组合作为我们的测试用例。这样测试用例的数量将大大减少到我们能够承受的范围之内。当然，我们需要一种方法来帮助我们做到这一点。这就是稍后要介绍的等价类划分法。

当有效输入（出）值域和无效输入（出）值域被进一步划分为更小的集合后，如何选定使用哪一些值作为代表来编写测试用例呢？这时就需要边界分析值法来帮助我们选取每个集合的关键值。

此外，在接下来的章节中我们还将介绍其他的黑盒测试方法以帮助我们在各种情况下写出合理的测试用例。这些方法应该被熟练掌握，然后在工作过程中被灵活运用。

2. 等价类划分（Equivalence Partitioning）

等价类划分运用的是对组件的输入值域和输出值域进行划分的模式来设计测试用例。我们一般可以通过对该组件的需求文档进行分析来得到这些输入值域和输出值域。当然，这时需求文档的深度、详细程度及精确程度等将对输入值域和输出值域分析都得产生巨大的影响。所以我们要求测试人员应当在需求文档编写期间积极参与文档的审查工作。

该设计模式应该包括对输入值域和输出值域的划分。每个被划分出来的等价类应当包含一组值的集合或一个范围之内的值。在每一个等价类中，所有的值对于这个组件来说在某种程度上都是等效（价）的。此外，有效值域和无效值域都应当按照这种方式来划分。

在这种设计模式下测试用例将执行（覆盖）各个等价类。这样的等价类包括对有效输入值域进行划分所得的等价类和对无效输入值域进行划分所得的等价类。测试用例还可以被设计为用来测试那些无法被推导出的无效输出值。

Offer

一个测试用例可以执行（覆盖）任意数量的等价类。每一个测试用例应当包括。

- 对于该组件的输入。
- 该用例所执行（覆盖）的等价类。
- 该用例的期望输出。

等价类划分设计模式的覆盖率为：

等价类划分覆盖率 =（已覆盖的等价类的数量/总的等价类数）×100%

在以上的公式中已覆盖的等价类的数量是指我们在测试用例设计过程中使用等价类划分设计模式所设计的测试用例集（test case suite）总共覆盖的等价类的数量。总的等价类数是指一个组件的输入值域和输出值域所有可能的等价类的总和。

说明

我们回忆一下，等价类划分基于以下的假设：根据组件的设计文档，该组件的输入值域和输出值域可以被划分为多个等价类。在每一个等价类中，所有的值对于这个组件来说在某种程度上都是等效的。因此，对一个等价类中的某一个值进行测试就可以被等效认为对这个等价类中所有的值进行了测试。

例子

假设有一个组件，为学生的成绩定级。它在其设计文档中被描述为以下内容：

该组件将接收两个有符号整型参数—考试成绩（80 分）和平时表现（20 分）。该组件将把这两个参数相加。之后根据这个和对其进行定级。经过该组件的运算和判断将输出 4 个值：

总分大于等于 80 - 'A'（优秀）

总分大于等于 60 并小于 80 - 'B'（良好）

总分大于等于 50 并小于 60 - 'C'（补考）

总分小于 50 - 'D'（重修）

在此，所有输入的考试成绩如果超出其有效值范围，则该组件将返回一个错误消息（error message）的提示信息。

对于这个例子，我们首先划分其等价类，然后再设计测试用例来执行（覆盖）每个等价类。等价类将按照该组件的输入和输出域来进行划分。输入域和输出域中所有的有效值域和无效值域都将被考虑。

下面先来分析该组件的输入域。

先对其两个参数的有效值域进行划分。

- 0≤考试成绩≤80。
- 0≤平时表现≤20。

然后对这两个参数的其无效值域进行划分。

- 考试成绩<0。

Offer!

- 考试成绩＞80。
- 平时表现＜0。
- 平时表现＞20。

当然我们还有可能碰到用户可能会误输入带小数的浮点数或一些字母、符号等非数字的字符组合。于是我们在这两个参数的输入域中还划分出以下的无效值域。

- 考试成绩=带小数的浮点数。
- 考试成绩=非数字的字符组合。
- 平时表现=带小数的浮点数。
- 平时表现=非数字的字符组合。

下面来分析一下该组件的输出域。

先对其输出域的有效值域进行划分：

- 80≤总分≤100　　将输出　‘A’（优秀）
- 60≤总分＜80　　将输出　‘B’（良好）
- 50≤总分＜60　　将输出　‘C’(补考)
- 0≤总分＜50　　将输出　‘D’（重修）
- 100＜总分　　将输出　‘error message’(错误消息)
- 总分＜0　　将输出　‘error message’(错误消息)

在此总分=考试成绩+平时表现。

我们注意到‘error message’(错误消息)也在输出域的有效值域之列。这是因为该组件的设计文档中已经指明在考试成绩和平时表现超出有效范围时有这种类型的输出。

以上6种情况之外的输出就是该输出域的无效输出了。一般来说未在组件的设计文档中描述的无效输出是较难被界定出来到的。但是一旦测试人员在使用该组件时遇到或界定出某一无效输出时就应当要求开发人员将其加入设计文档并在新的组件版本中将其实现。

正因为无效输出很难被界定，所以设计无效输出的测试用例就要求测试人员有较强的与被测软件相关的业务背景和较多的经验。因此，测试用例设计人员对于无效输出域进行的等价类的划分都具有主观性。对于这个例子，我们将其输出域的无效值域划分为以下等效类。

- output = 'null' 等效于输入的两个参数皆为 null。
- output = 'float input' 等效于输入参数至少有一个是浮点数。
- output = 'non-numeric input' 等效于输入参数至少有一个是非数字。

下面来总结一下所有等价类的划分。

- 0≤考试成绩≤80。
- 0≤平时表现≤20。
- 考试成绩＜0。
- 考试成绩＞80。
- 平时表现＜0。



- 平时表现＞20。
- 考试成绩=带小数的浮点数。
- 考试成绩=非数字的字符组合。
- 平时表现=带小数的浮点数。
- 平时表现=非数字的字符组合。
- 80≤总分≤100　　将输出　‘A’（优秀）。
- 60≤总分＜80　　将输出　‘B’（良好）。
- 50≤总分＜60　　将输出　‘C’(补考)。
- 0≤总分＜50　　将输出　‘D’（重修）。
- 100＜总分　　将输出　‘error message’(错误消息)。
- 总分＜0　　将输出　‘error message’(错误消息)。
- output = 'null' 等效于输入的两个参数皆为 null。
- output = 'float input' 等效于输入参数至少有一个是浮点数。
- output = 'non-numeric input' 等效于输入参数至少有一个是非数字。

下面将设计测试有例来覆盖以上的各个等价类，此时我们将采取逐个覆盖的方法。这样可以使我们设计出的用例和被其覆盖的等价类之间的关系显得非常清晰。

以下用例是对与输入参数-考试成绩相关的等价类进行覆盖，如表 3-3 所示。

表 3-3　　输入参数-考试成绩相关的等价类

测试用例	001	002	003
输入参数 1(考试成绩)	50	-3	85
输入参数 2(平时表现)	15	15	15
总分	65	12	100
被覆盖的划分（考试成绩相关）	0≤考试成绩≤80	考试成绩＜0	u 考试成绩＞80
输出结果	B	error message	error message

注：其中**平时表现**被设定为任意的有效值 15。

以下用例是对与输入参数-平时表现相关的等价类进行覆盖，如表 3-4 所示。

表 3-4　　输入参数-平时表现相关的等价类

测试用例	004	005	006
输入参数 1（考试成绩）	70	70	70
输入参数 2（平时表现）	18	-8	25
总分	88	62	95
被覆盖的划分（平时表现相关）	0≤平时表现≤20	平时表现＜0	平时表现＞20
输出结果	A	error message	error message

注：其中**考试成绩**被设定为任意的有效值 70。

Offer

以下用例是对与输入参数的其他无效值域相关的等价类进行覆盖，如表3-5所示。

表3-5 输入参数的其他无效值域相关的等价类

测试用例	007	008	009	010
输入参数1(考试成绩)	70.1	a	70	70
输入参数2(平时表现)	15	15	12.5	good
总分	85.1	n/a	72.5	n/a
被覆盖的划分	考试成绩=带小数的浮点数	考试成绩=非数字的字符组合	平时表现=带小数的浮点数	平时表现=非数字的字符组合
输出结果	float input*	non-numeric input*	float input*	non-numeric input*

注：以上带*的**输出结果**都未在组件的设计文档中详细描述。

以下用例是对与输出域的有效值域的等价类进行覆盖，如表3-6所示。

表3-6 输出域的有效值域的等价类

测试用例	011	012	013
输入参数1(考试成绩)	-5	20	45
输入参数2(平时表现)	-2	10	10
总分	-7	30	55
被覆盖的划分	总分＜0	0≤总分＜50	50≤总分＜60
输出结果	error message	D	C
测试用例	014	015	016
输入参数1(考试成绩)	50	75	85
输入参数2(平时表现)	16	15	25
总分	66	90	110
被覆盖的划分	60≤总分＜8	80≤总分≤100	100＜总分
输出结果	B	A	error message

以下用例是对与输出域的无效值域的等价类进行覆盖，如表3-7所示。

表3-7 输出域的无效值域的等价类

测试用例	017	018	019
输入参数1(考试成绩)	12.4	ww	null
输入参数2(平时表现)	16	as	null
总分	18.4	90	null
被覆盖的划分	输入参数中带小数的浮点数	输入参数中非数字的字符组合	null
输出结果	float input*	non-numeric input*	null*

注：以上带*的**输出结果**都未在组件的设计文档中详细描述。

Offer

回顾我们所设计的用例不难发现，有许多用例都比较相似。例如 007 和 017，不同的只是它们被设计的目的是为了覆盖不同的等价类。仔细观察测试用例我们还可以发现以上的每个测试用例都涵盖了两个输入值和一个输出值。这就意味着每个测试用例实际上覆盖了 3 个等价类：两个输入域的等价类和一个输出域的等价类，这也就意味着我们可以用更小的测试用例集合来覆盖我们所划分的所有的等价类。下面就来尝试缩小以上测试用例的集合，如表 3-8 所示。

表 3-8 缩小以上测试用例的集合

测试用例	001	002	003	004
输入参数 1(考试成绩)	76	60	45	30
输入参数 2(平时表现)	16	15	10	10
总分	92	75	55	40
被覆盖的划分(考试成绩)	0≤考试成绩≤80	0≤考试成绩≤80	0≤考试成绩≤80	0≤考试成绩≤80
被覆盖的划分(平时表现)	0≤平时表现≤20	0≤平时表现≤20	0≤平时表现≤20	0≤平时表现≤20
被覆盖的划分(输出)	A	B	C	D
测试用例	005	006	007	008
输入参数 1(考试成绩)	-5	90	45.5	we
输入参数 2(平时表现)	-6	30	10.6	our
总分	-11	120	56.1	n/a
被覆盖的划分(考试成绩)	考试成绩＜0	100＜考试成绩	考试成绩=带小数的浮点数	考试成绩=非数字的字符组合
被覆盖的划分(平时表现)	平时表现＜0	20＜平时表现	平时表现=带小数的浮点数	平时表现=非数字的字符组合
被覆盖的划分(输出)	error message	error message	float input*	non-numeric input*
测试用例	009			
输入参数 1(考试成绩)	null			
输入参数 2(平时表现)	null			
总分	null			
被覆盖的划分(考试成绩)	null			
被覆盖的划分(平时表现)	null			
被覆盖的划分(输出)	null*			

注：以上带*的**输出结果**都未在组件的设计文档中详细描述。

我们可以看到，以上两种用以覆盖等价类的测试用例设计方法（逐个覆盖法和最小集合法）都能做到覆盖所有的等价类。不同的是，逐个覆盖法需要更多的测试用例。而运用最小集合法时，如果测试用例全部通过，则等效于所有的等价类都顺利地通过了测试，但是一旦出现了某一个或多个测试用例未通过的情况，我们就必须设计和运行更多的测试用例以定位出现缺陷的等价类。

通常来说，分析组件的设计文档、界定并划分等价类的时间远高于产生测试用例和执行测



试用例所花的时间，所以如果经过初步分析，最小集合法不能大量减少测试用例，则一般都直接采用逐个覆盖法所产生的测试用例的集合。

3. 边界值分析（Boundary Value Analysis）

边界值分析法运用的是将被测组件的输入值域和输出值域划分为多个有序并有确定边界的集合的模式来设计测试用例。我们一般可以通过对该组件的需求文档进行分析来得到这个输入值域和输出值域。

在这个模式下，被划分出来的集合是有序的，而且包含明确的边界。每个被划分出来的集合应当包含一组值或一个取值的范围。在每一个被划分出的集合中，所有的值对于这个组件来说在某种程度上都是等效（价）的。此外，有效值域和无效值域都应当按照这种方式来划分。

所谓边界值就是用来界定各个有序集合之间的边界点。每个被划分出来的集合一般都有一个最大值和一个最小值作为其边界值，而其他的值都在这两个边界值的范围之内。

在边界值分析法的模式下设计出来的测试用例将用于覆盖每个被划分出来的集合的边界点上的值和其相邻值（边界点的前一个有序值和边界点的后一个有序值）。因此，对于每一个被确定的边界都有3个值需要覆盖。这样的边界值是对有效输入值域进行划分所得的集合的边界值和对无效输入值域进行划分所得的集合的边界值。另外，测试用例还可以被设计为用来测试那些无法被推导出的无效输出值域的边界值。

每一个运用边界值分析法模式设计的测试用例都应当包括。

- 对于该组件的输入。
- 该用例所执行（覆盖）的边界值。
- 该用例的期望输出。

边界值分析设计模式的覆盖率为：

边界值覆盖率=(被执行的已确定边界值的数量/总的边界值数)×100%

以上公式中每一个边界值均对应于测试用例中所覆盖的边界值或其相邻值。

说明

边界值分析是基于以下假设。根据组件的设计文档，该组件的输入值域和输出值域可以被划分为多个集合（通常情况下我们会把从输入值域和输出值域划分出来的等价类作为需要收集边界点的集合）。操作这些集合的边界点上的边界值及其相邻值对于开发人员来说是最容易犯错误的。基于这个模式下，我们设计的测试用例就是用来覆盖这些边界。下面将继续使用上一节中的设计文档来设计边界值分析模式下的测试用例。

例子

假设有一个组件，为学生的成绩定级。它在其设计文档中被描述为以下内容：

该组件将接收两个有符号整型参数—考试成绩（80分）和平时表现（20分）。该组件把这两个参数相加，之后根据该值对其进行定级。经过该组件的运算和判断将输出4个值：

总分大于等于80 - ‘A’（优秀）

Offer!

总分大于等于 60 并小于 80 - 'B'（良好）
总分大于等于 50 并小于 60 - 'C'（补考）
总分小于 50 - 'D'（重修）
在此，所有输入的考试成绩如果超出其有效值范围，则该组件将返回一个错误消息（error message）的提示信息。

对于这个例子，我们首先划分其等价类，然后再确定各个等价类的边界，并设计测试用例覆盖它们。此时等价类将按照该组件的输入和输出域来进行划分。输入域和输出域中所有的有效值域和无效值域都将被考虑。

下面来列举该组件的输入域。

首先对其两个参数的有效值域进行划分。

- 0≤考试成绩≤80。
- 0≤平时表现≤20。

然后对这两个参数的其无效值域进行划分。

- 考试成绩＜0。
- 考试成绩＞80。
- 平时表现＜0。
- 平时表现＞20。

图 3-7 所示是组件输入值域各个等价类的边界点值及其相邻值。

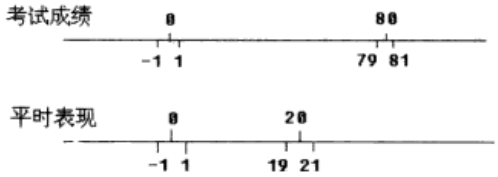


图 3-7 组件输入值域各个等价类的边界点值及其相邻值

由此可以看到，在边界值分析设计模式下我们在每一个被划分的等价类的边界点附近取 3 个值：边界点的前一个有序值、边界点值、边界点的后一个有序值。

因此，对于考试成绩即可设计以下 6 个用例，如表 3-9 所示。

表 3-9 考试成绩测试用例

测试用例	001	002	003	004	005	006
输入参数 1(考试成绩)	-1	0	1	79	80	81
输入参数 2(平时表现)	15	15	15	15	15	15
总分	14	15	16	94	95	96
被覆盖的边界值(考试成绩相关)	0			80		
输出结果	error message	D	D	A	A	error message

注：其中**平时表现**被设定为任意的有效值 15。

而对于平时表现就可以设计以下6个用例，如表3-10所示。

表3-10 设计6个用例

测试用例	007	008	009	010	011	012
输入参数1(考试成绩)	75	75	75	75	75	75
输入参数2(平时表现)	-1	0	1	19	20	21
总分	74	75	76	94	95	96
被覆盖的边界值(平时成绩相关)	0			20		
输出结果	error message	B	B	A	A	error message

注：其中**考试成绩**被设定为任意的有效值70。

当然，该组件的输入值域还可以划分出以下的无效值域。

- 考试成绩=带小数的浮点数。
- 考试成绩=非数字的字符组合。
- 平时表现=带小数的浮点数。
- 平时表现=非数字的字符组合。

由于上述无效值域的等价类并不具备“有序”的特征，而且在这些等价类中也很难确定其边界点，所以在边界值分析设计模式下将不对这些等价类设计测试用例。

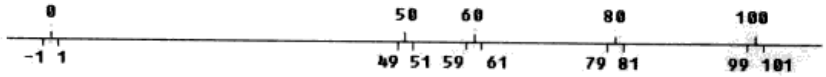
下面来列举一下输出值域等价类。

- 80≤总分≤100 将输出 ‘A’（优秀）
- 60≤总分＜80 将输出 ‘B’（良好）
- 50≤总分＜60 将输出 ‘C’（补考）
- 0≤总分＜50 将输出 ‘D’（重修）
- 100＜总分 将输出 ‘error message’（错误消息）
- 总分＜0 将输出 ‘error message’（错误消息）

注：在此总分=考试成绩+平时表现。

“‘error message’(错误消息)”也应在输出域的有效值域之列。这是因为在该组件的设计文档中已经指明在考试成绩和平时表现超出有效范围时有这种类型的输出。

下图所示为各个该组件输出值域各个等价类的边界点值及其相邻值。



因此，对于输出值域就可以设计以下用例，如表3-11所示。

表3-11 对于输出值域设计用例

测试用例	013	014	015	016	017	018
输入参数1(考试成绩)	0	0	1	29	25	40
输入参数2(平时表现)	-1	0	0	20	25	11

Offer

续表

测试用例	013	014	015	016	017	018
总分	-1	0	1	49	50	51
被覆盖的输出值域边界值	0			50		
输出结果	error message	D	D	D	C	C
测试用例	019	020	021	022	023	024
输入参数 1(考试成绩)	39	49	51	65	72	66
输入参数 2(平时表现)	20	11	10	14	8	15
总分	59	60	61	79	80	81
被覆盖的输出值域边界值	60			80		
输出结果	C	B	B	B	A	A

测试用例	025	026	027
输入参数 1(考试成绩)	79	80	81
输入参数 2(平时表现)	20	20	20
总分	99	100	101
被覆盖的输出值域边界值	100		
输出结果	A	A	error message

对于在上一节对输出值域所划分的以下无效值等价类，因为其也不具备“有序”的特征，所以在这些等价类中很难确定其边界点。因此在边界值分析设计模式下将不对这些等价类设计测试用例。

- output = 'null' 等效于输入的两个参数皆为 null。
- output = 'float input' 等效于输入参数至少有一个是浮点数。
- output = 'non-numeric input' 等效于输入参数至少有一个是非数字。

然而除了以上被划分的等价类的边界值外还有一组隐性的边界值未确定。当仔细分析该组件的设计文档时不难发现，文档中规定两个输入参数是有符号整型。文档中没有规定存储总分的变量是什么类型，此时不妨假设开发人员把它也设为有符号整型。而计算机上的有符号整型是有取值范围的。假设这个程序在普通的 32 位机上运行，那么 32 位机上有符号整型的取值范围是-2^{31}～2^{31}-1。因此就有：

- 考试成绩＜0　　等价于　-231＜考试成绩＜0
- 考试成绩＞80　　等价于　80＜考试成绩＜231-1
- 平时表现＜0　　等价于　-231＜平时表现＜0
- 平时表现＞20　　等价于　20＜平时表现＜231-1
- 100＜总分　　等价于　100＜总分＜231-1
- 总分＜0　　等价于　-231＜总分＜0

因此我们设计了以下用例来覆盖这些隐性的边界值，如表3-12所示。

表3-12 设计了以下用例来覆盖这些隐性的边界值

测试用例	028	029	030	031	032	033
输入参数1(考试成绩)	$-2^{31}-1$	-2^{31}	$-2^{31}+1$	$2^{31}-2$	$2^{31}-1$	2^{31}
输入参数2(平时表现)	20	20	20	14	8	15
被覆盖的边界值(考试成绩相关)	-2^{31}			$2^{31}-1$		
输出结果	error message	error message	error message	error message	error message	error message
测试用例	034	035	036	037	038	039
输入参数1(考试成绩)	70	70	70	70	70	70
输入参数2(平时表现)	$-2^{31}-1$	-2^{31}	$-2^{31}+1$	$2^{31}-2$	$2^{31}-1$	2^{31}
被覆盖的边界值(平时成绩相关)	-2^{31}			$2^{31}-1$		
输出结果	error message	error message	error message	error message	error message	error message
测试用例	040	041	042	043	044	045
输入参数1(考试成绩)	-2^{30}	-2^{30}	$-2^{30}+1$	$2^{30}-1$	2^{30}	2^{30}
输入参数2(平时表现)	$-2^{30}-1$	-2^{30}	-2^{30}	$2^{30}-1$	$2^{30}-1$	2^{30}
总分	$-2^{31}-1$	-2^{31}	$-2^{31}+1$	$2^{31}-2$	$2^{31}-1$	2^{31}
被覆盖的输出值域边界值	-2^{31}			$2^{31}-1$		
输出结果	error message	error message	error message	error message	error message	error message

对于该组件来说，以上45条用例100%地覆盖了其输入值域和输出值域的边界点的值及其相邻值 。

4. 状态转换测试法（State Transition Testing）

状态转换测试法应用于以下模式：被测组件拥有多个状态(state)，各个状态之间的转换（transition）由事件(event)来触发，各个状态之间的转换还可能导致一些动作（action）的产生。在该种模式下被测组件应当包括状态、转换、事件、动作及它们之间的联系。其中各个状态之间是正交（disjoint）的。各个状态是可以被明确识别的，而且其数量是有限的。事件用于触发各状态之间的转换。而事件的触发将由对该组件的输入产生。作为结果，状态的转换可能导致一些动作的产生，从而可能进一步产生输出。

在该种模式下通常用状态转换图、状态转换模型或状态转换表来描述被测组件各状态之间的转换。

Offer

在该种设计模式下测试用例将执行（覆盖）各个状态之间的转换。每条测试用例可以覆盖多个转换。但每条用例必须包括。

- 该组件的起始状态。
- 对该组件的输入。
- 期望输出。
- 期望的最终状态。

对于每条测试用例中的期望状态，我们必须详细描述以下内容。

- 起始状态。
- 触发该状态转换至下一状态的事件。
- 该转换所产生的期望动作。
- 转换后的期望状态。

在这种设计模式下测试用例可以被设计用来测试各状态之间的有效转换，还可以被设计用来测试那些未在组件测试文档中明示的转换。

该黑盒技术是基于被测组件涉及状态的转换为模式。下面将通过一个实例来描述如何使用此方法。

例子

该组件为一个手机上流媒体播放软件的显示屏幕。它有 4 种状态的显示："显示正在播放的节目"（BF），"暂停时显示正在播放帧的静止画面"（JZ），"停止时显示节目开头第一帧画面"（TZ）和"无信号时显示'无信号'提示画面"（WXH）。

触发状态转换的输入有："单击停止按钮"，"单击播放按钮"，"单击暂停按钮"，"屏蔽信号"，"信号恢复"。

状态转换如下：

"单击停止按钮"可以使显示屏幕从"显示正在播放的节目"或"显示正在播放帧的静止画面"状态转换为"显示节目开头第一帧画面" 状态。

"单击暂停按钮"可以使显示屏幕从"显示正在播放的节目"状态转换为"显示正在播放帧的静止画面" 状态。

"单击播放按钮"可以使显示屏幕从"显示节目开头第一帧画面"或"显示正在播放帧的静止画面"状态转换为"显示正在播放帧的静止画面"。

"屏蔽信号"可以使显示屏幕从"显示正在播放的节目"状态转换为"无信号时显示'无信号'提示图片"。

"信号恢复"可以使显示屏幕从"无信号时显示'无信号'提示图片" 状态转换为"显示正在播放的节目"。

Offer

图 3-8 所示为一个组件由事件触发而从一个状态转换到另一个状态，并产生输出。

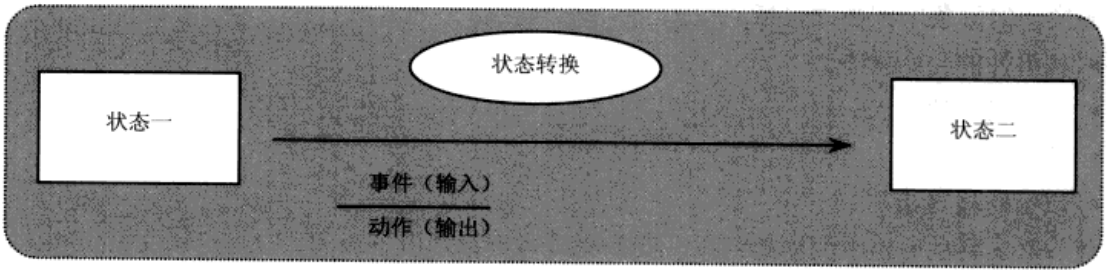


图 3-8 一个组件由事件触发而从一个状态转换到另一个状态

因此，根据上例组件设计文档中的描述，我们可将图 3-7 转换为如图 3-9 所示。

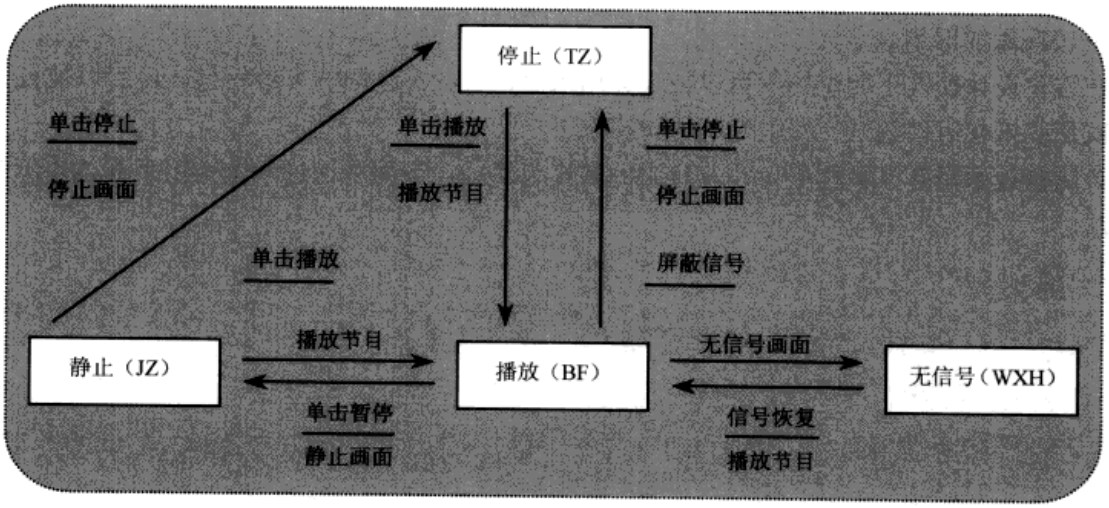


图 3-9 转化后的结果

下面将图 3-8 转换为下列表格，如表 3-13 所示。

表 3-13 转换后的表格

测试用例	001	002	003	004	005	006	007
起始状态	JZ	TZ	BF	BF	JZ	BF	WXH
事件（输入）	单击停止	单击播放	单击停止	单击暂停	单击播放	屏蔽信号	信号恢复
动作（输出）	停止画面	播放节目	停止画面	静止画面	播放节目	无信号画面	播放节目
结束状态	TZ	BF	TZ	JZ	BF	WXH	BF

注：上表中，BF 表示“显示正在播放的节目”状态，JZ 表示“暂停时显示正在播放帧的静止画面”，TZ 表示“停止时显示节目开头第一帧画面”， WXH 表示“无信号时显示‘无信号’提示画面”。

这一组 7 条测试用例覆盖了该组件设计文档中所描述的所有状态转换，但是这组用例都只执行了一个状态的转换，因此这一级别的覆盖方式也被称为 0-switch 覆盖（最基础的覆盖）。这

Offer

样的用例可以测出那些比较显而易见的错误转换或错误输出，但是这样的用例无法测试到需要进行一系列状态转换才能出现的问题。因此可以提高我们的覆盖等级，将测试用例设计为1-switch覆盖以在一个测试用例中执行多个状态的转换。

在设计该组件的每一个1-switch覆盖的测试用例时，从4个状态中的某一个状态出发，然后进行两次状态的转换并覆盖所有可能的路径。结果如表3-14和表3-15所示。

表3-14 进行两次状态的转换并覆盖所有可能的路径（一）

测试用例	008	009	010	011	012	013	014
起始状态	WXH	WXH	WXH	BF	BF	BF	BF
事件（输入）	单击播放	信号恢复	信号恢复	屏蔽信号	单击停止	单击暂停	单击暂停
动作（输出）	播放节目	播放节目	播放节目	无信号画面	停止画面	静止画面	静止画面
下一状态	BF	BF	BF	WXH	TZ	JZ	JZ
事件（输入）	屏蔽信号	单击停止	单击暂停	信号恢复	单击播放	单击播放	单击停止
动作（输出）	无信号画面	停止画面	静止画面	播放节目	播放节目	播放节目	停止画面
最终状态	WXH	TZ	JZ	BF	BF	BF	TZ

表3-15 进行两次状态的转换并覆盖所有可能的路径（二）

测试用例	015	016	017	018	019	020	021
起始状态	TZ	TZ	TZ	JZ	JZ	JZ	JZ
事件（输入）	单击播放	单击播放	单击播放	单击停止	单击播放	单击播放	单击播放
动作（输出）	播放节目	播放节目	播放节目	停止画面	播放节目	播放节目	播放节目
下一状态	BF	BF	BF	TZ	BF	BF	BF
事件（输入）	单击停止	屏蔽信号	单击暂停	单击播放	单击暂停	单击停止	屏蔽信号
动作（输出）	停止画面	无信号画面	静止画面	播放节目	静止画面	停止画面	无信号画面
最终状态	TZ	WXH	JZ	BF	JZ	TZ	WXH

注：上表中，BF表示“显示正在播放的节目”状态，JZ表示“暂停时显示正在播放帧的静止画面”，TZ表示“停止时显示节目开头第一帧画面”，WXH表示“无信号时显示‘无信号’提示画面”。

以上就是1-switch覆盖的测试用例。可以看到提高了覆盖等级之后每个测试用例覆盖了两个状态的转换，因此也可以依照上述的方法把覆盖等级提高到n-switch覆盖。这样，则每个测试用例可以覆盖n个状态的转换。也就是说，可以测试到一些更深层次的、需要进行更多状态转换才能出现的问题。

仔细观察以上1-switch覆盖的测试用例不难发现，所有1-switch覆盖测试用例中的状态转换都已被包含在0-switch覆盖的测试用例中。实际上0-switch覆盖的测试用例中的状态装换是所有n-switch覆盖的测试用例的状态转换的全集，但0-switch覆盖的测试用例中的状态转换一般只局限于组件设计文档中的描述。一般情况下，组件设计文档可能不会显式地描述一些隐藏的状态转换或一些无效的状态转换。这也就意味着0-switch覆盖乃至于n-switch覆盖的测试用

Offer

例都可能会遗漏这一类隐藏的状态转换或无效的状态转换。

如何找到组件设计文档中未显示的描述一些隐藏的状态转换或一些无效的状态转换呢？下面可以通过使用一种被称作状态表（State Table）的表格来实现，如表 3-16 所示。

表 3-16 状态表（State Table）的表格

	输入 1	输入 2	输入 3	……
状态 1	状态/输出对 A	状态/输出对 B	状态/输出对 C	……
状态 2	状态/输出对 D	状态/输出对 E	状态/输出对 F	……
状态 3	状态/输出对 G	状态/输出对 H	状态/输出对 I	……
……	……	……	……	……

下面列出该组件的状态表，如表 3-17 所示。

表 3-17 组件的状态表

	点击播放	点击停止	点击暂停	屏蔽信号	信号恢复
BF	NA-1	TZ/停止画面	ZT/ 静止画面	WXH/无信号画面	NA-2
JZ	BF/播放节目	TZ/停止画面	NA-3	NA-4	NA-5
TZ	BF/播放节目	NA-6	NA-7	NA-8	NA-9
WXH	NA-10	NA-11	NA-12	NA-13	BF/播放节目

注：上表中，BF 表示“显示正在播放的节目”状态，JZ 表示“暂停时显示正在播放帧的静止画面”，TZ 表示“停止时显示节目开头第一帧画面”， WXH 表示“无信号时显示‘无信号’提示画面”，NA 表示未在组件设计文档中描述或该状态变换无效。

在上述状态表中可以看到有 4 种状态（BF、JZ、TZ、WXH）和 5 种输入（点击播放、点击停止、点击暂停、屏蔽信号、信号恢复），因此有 20（4×5）种状态/输出对的组合。如表 3-17 所示，这 20 种组合中有 7 种组合是在组件设计文档中被详细描述过的，正好对应于 0-switch 覆盖的测试用例中的所有状态变换。其余的 13 种组合被标上了 NA，意味着其未在组件设计文档中描述或该状态变换无效。例如 NA-4，在“暂停时显示正在播放帧的静止画面”的状态下对其进行“屏蔽信号”，这时的组件设计者可以选择“媒体播放软件的显示屏幕”显示“静止画面”或“无信号时显示‘无信号’提示画面”，但这并没有在设计文档中被明确描述。

强调

此类的状态转换（NA-4）就属于其本身可以有符合逻辑的解释，但由于其并不属于显而易见的转换。因此对其行为的定义容易被设计人员忽略。而对于软件人我们就有义务把这些被忽略的点找出来，并将其提交给设计人员敦促其给出详细的定义，然后再对新的设计文档重新画出状态转换图，重新分析，并重新设计测试用例。

另外，例如 NA-13（在“无信号时显示‘无信号’提示画面”状态下“屏蔽信号”）等，就

Offer

属于无效状态变换，可以将其排除，并不对其设计测试用例。

5. 因果图法（Cause Effect Graphing）

因果图法应用于以下模式：被测组件多组因果关系的逻辑组合，每一个原因被表示为包含逻辑真/假的输入或输入组合的条件，每一个结果被表示为包含输出或输出组合的布尔表达式。在该模式下，组件被描述为一个逻辑图。测试人员可以基于这张逻辑图开发出一张包含因果关系判的定表。

依照该模式设计的测试用例应当包含定义该组件各个输入与输出之间关系的规则。每一个规则都对应于一个惟一的一组输入的组合。每一个测试用例应当包括。

- 每一个原因的布尔状态（真/假值）。
- 每一个结果的布尔状态。

因果图法的覆盖率=（被执行的规则的数量/规则的总数）×100%

下面将通过一个实例来描述如何使用此方法。

例子

该组件用于处理信用卡刷卡时对不同数额的支付款项进行扣款，并输出扣款后的结果。

持卡人分短信用户和非短信用户。短信用户的信用卡账户在短信通知限额内扣款（<信用限额），扣款将被执行，持卡人将不会收到银行的短信通知。短信用户账户在超出短信通知限额时扣款，扣款将被执行，银行将发送短信通知持卡人。短信用户和非短信用户的信用卡账户在扣款额高于信用额度时，扣款将不被执行，账户被挂起。

图 3-10 所示为因果的图例。

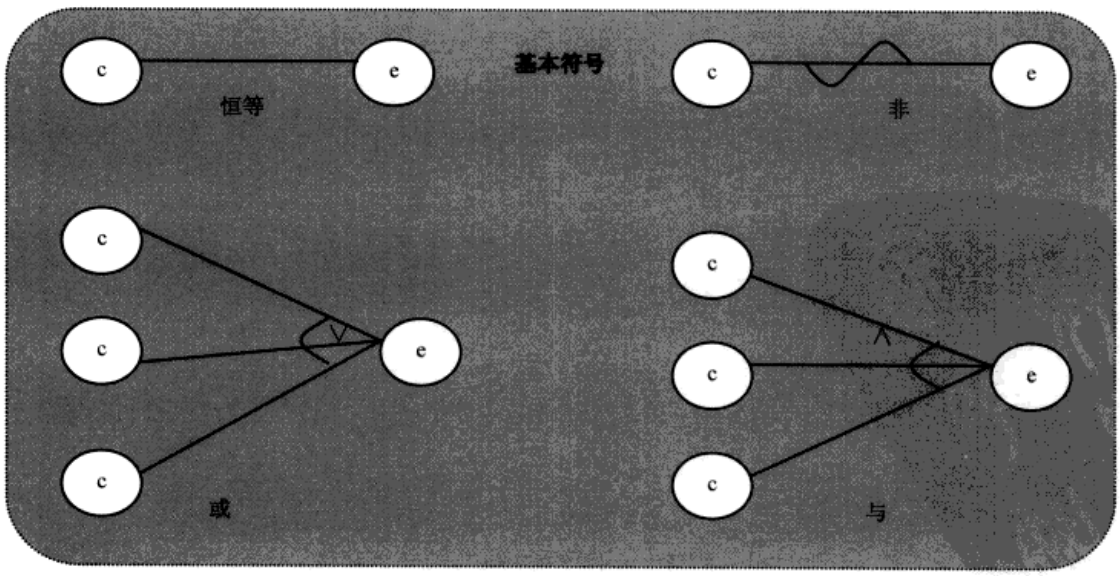


图 3-10 因果图的图例

Offer

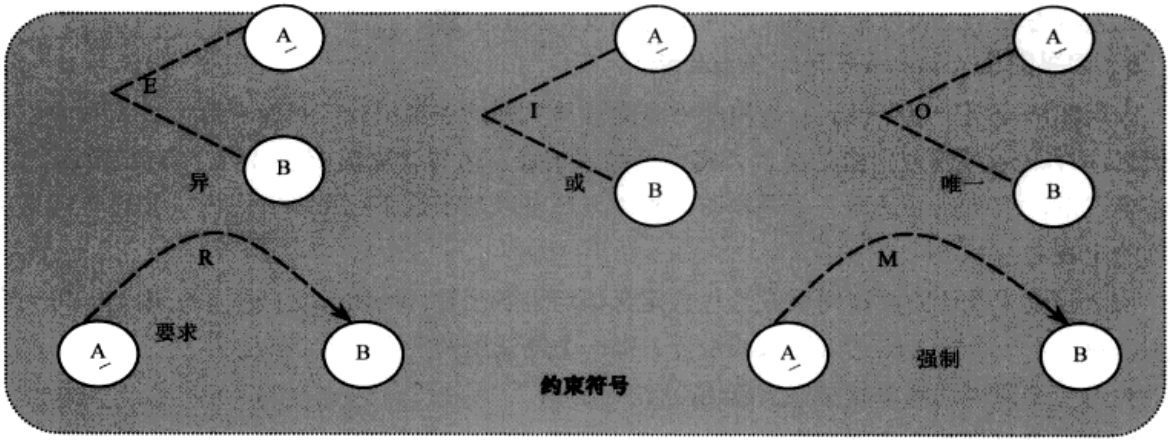


图 3-10 因果图的图例（续）

根据例题中的组件需求文档，我们可以列出以下原因和结果。

原因。

- c1 短信通知限额内扣款。
- c2 短信通知限额外，但信用额度内扣款。
- c3 短信用户。

结果。

- e1 执行扣款。
- e2 账户挂起。
- e3 银行向用户发送短信。

图 3-11 所示为该组件所对应的因果图。

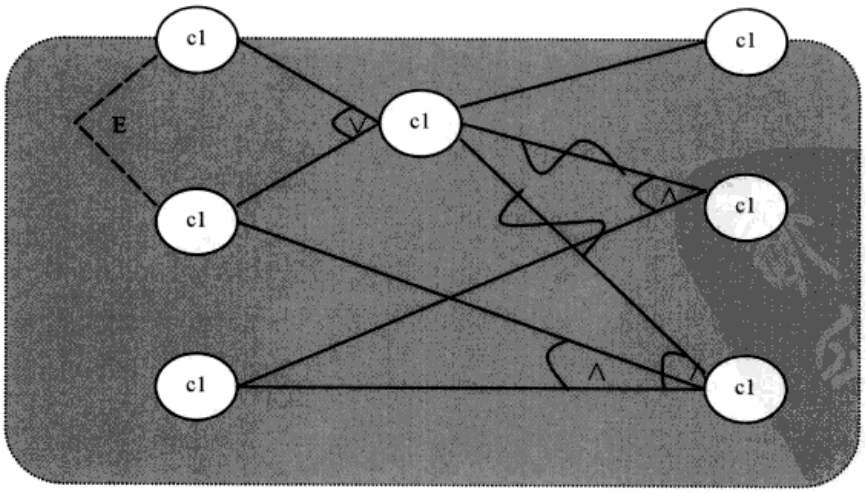


图 3-11 组件所对应的因果图

表 3-18 所示为因果图所对应的判定表。

表 3-18 因果图所对应的判定表

	1	2	3	4	5	6	7	8
c1	T	T	F	F	F	F	T	T
c2	F	F	T	T	F	F	T	T
c3	F	T	T	F	F	T	T	F
e1	T	T	T	T	F	F	NA	NA
e2	F	F	F	F	T	T	NA	NA
e3	F	F	T	F	F	T	NA	NA

此后我们就可以根据表 3-18 所示的判定表写出测试用例了。读者可以尝试自行完成。

6. 语法测试（Syntax Test）

语法测试运用于某些组件模型，这些组件的输入值有正式定义的语法。语法本身代表的是一系列的规则，而每条规则规定了一个或一些标记在输入值中所处的位置、出现的频率以及各标记之间的关系。我们可以对该类组件的输入值的语法规则进行分析从而设计出对该输入值域的有效值域（符合正式语法）和无效值域（不符合正式语法）进行覆盖的测试用例。

用于覆盖有效值域（符合正式语法）的测试用例应当执行各语法元素规则所派生出来的各个选项分支。该类测试用例应当设计如下。

- 当语法元素含有多个可选择的值时，该语法元素的所有选项都必须有用例覆盖。
- 当语法元素可以重复出现时，测试用例中至少要覆盖以下两个选项：可重复出现的语法元素出现最少次数的情况和可重复出现的语法元素出现超过最少次数的情况。

一个测试用例可以执行上述的多个选项。每一个测试用例包括以下部分。

- 对组件的输入。
- 测试用例所执行的选项。
- 测试用例的期望输出。

用于覆盖无效值域（不符合正式语法）的测试用例应当设计如下。

- 设计一张体现语法元素产生常见错误的变体的列表。
- 该变体表中的变化可应用于语法元素以便于产生覆盖无效值域（不符合正式语法）的测试用例。
- 设计测试用例来覆盖所有变体/语法元素组合。
- 应当设计一些测试用例用于覆盖特别的变化。

对于每一个覆盖无效值域（不符合正式语法）的测试用例应当包括以下部分。

- 对组件的输入。
- 被执行的变化。
- 被执行变化所覆盖的语法元素。
- 测试用例的期望输出。

Offer!

下面将通过一个实例来描述如何使用此方法。

例子

该组件用于将一个被输入的字符串转化成一个浮点数。

下面采用 BNF（Backus Naur Form）范式来表示浮点数的语法。

先来熟悉一下 BNF 范式的一些符号的定义。

- 在双引号中的字（“word”）代表着这些字符本身。
- 在双引号外的字（有可能有下划线）代表着语法部分。
- 尖括号（ <>）内包含的为必选项。
- 方括号（ [] ）内包含的为可选项。
- 大括号（ { } ）内包含的为可重复 0 至无数次的项。
- 竖线（ | ）表示在其左右两边任选一项，相当于“OR”的意思。
- ::= 是“被定义为”的意思。

浮点数的语法可以被表示为。

- 浮点数::=<整数>[<小数点><数字组合>]。
- 整数::==[符号]<无符号整数>。
- 符号::=< “+” >|< “-” >。
- 无符号整数::=（<非零数>[数字组合]）|<零>。
- 小数点::=< “.” >。
- 数字组合::={<数字>}。
- 数字::=<零>|<非零数>。
- 零::=< “0” >。
- 非零数::=< “1” >|< “2” >|< “3” >|< “4” >|< “5” >|< “6” >|< “7” >|< “8” >|< “9” >。

有效输入。

由表 3-18 可以得出。

- 浮点数中有两个选择项：<整数>opt1, <整数>[<小数点><数字组合>]opt2。
- 整数中有两个选项：<无符号整数>opt3, [符号]<无符号整数>opt4。
- 符号中有两个选项：< “+” >opt5,< “-” >opt6。
- 无符号整数有 3 个选项：<非零数>[数字组合]opt7, <非零数>opt8, <零>opt9。
- 小数点无选项。
- 数字组合有两个选项：单个<数字>opt10 或多个<数字>opt11。
- 数字有两个选项：<零>opt12，<非零数>opt13。
- 零无选项。
- 非零数有 9 个选项：每个非零数各为一个选项，我们把它记为 opt14-opt22。

因此有 21 个选项需要覆盖。下面将设计测试用例来覆盖这 21 个选项，如表 3-19 所示。

Offer

表 3-19 设计测试用例来覆盖这 21 个选项

测试用例	被输入浮点数	被覆盖的选项	测试范围
1	12	opt1	有效输入
2	3.14	opt2	有效输入
3	23	opt3	有效输入
4	33	opt4	有效输入
5	43.6	opt5	有效输入
6	-54.43	opt6	有效输入
7	1787	opt7	有效输入
8	6	opt8	有效输入
9	0.1	opt9	有效输入
10	2.1	opt10	有效输入
11	4.123457	opt11	有效输入
12	0	opt12	有效输入
13	1.1	opt13	有效输入
14	1.1	opt14	有效输入
15	2.2	opt15	有效输入
16	3.3	opt16	有效输入
17	4.4	opt17	有效输入
18	5.5	opt18	有效输入
19	6.6	opt19	有效输入
20	7.7	opt20	有效输入
21	8.8	opt21	有效输入
22	9.9	opt22	有效输入

当然，以上的测试用例集合并不是最小集合。我们不难发现有些测试用例实际上可以覆盖多个“有效输入”，因此读者可以自行尝试将其缩减。

无效输入。

首先构建一个总的变体列表。

- 变体 1：引入语法元素的无效值。
- 变体 2：用其他已定义的语法元素来替代一个元素。
- 变体 3：遗漏一个已定义的语法元素。
- 变体 4：添加一个额外的元素。

下面将所有变体应用于每一个语法元素，用 e_X 来代替的浮点数定义中的每个语法元素。

- 浮点数：：=<整数>[<小数点><数字组合>]

e_1：：=e_2e_3e_4

Offer

• 整数:: ==[符号]<无符号整数>

e_5:: =e_6e_7

• 符号:: =<“+”>|<“-”>

e_8:: =e_9

• 无符号整数:: =(<非零数>[数字组合])|<零>

e_10:: =e_11e_12e_13

• 小数点:: =<“.”>

e_14:: =e_15

• 数字组合:: ={<数字>}

e_16:: =e_17

• 数字:: =<零>|<非零数>

e_18:: =e_19e_20

• 零:: =<“0”>

e_21:: =e_22

• 非零数:: =<“1”>|<“2”>|<“3”>|<“4”>|<“5”>|<“6”>|<“7”>|<“8”>|<“9”>

e_23:: =e_24

表3-20就是根据所有语法元素的不同变体所设计的覆盖无效输入的测试用例，其中省略了一些与表中用例重复的变体/语法用例的组合。当然还可以将多个变体应用于每个语法元素上来设计更多的测试用例。

表3-20 根据所有语法元素的不同变体所设计的覆盖无效输入的测试用例

测试用例	被输入浮点数	被覆盖的变体	被覆盖的语法元素	测试范围
1	a.3	变体1	e_2	无效输入
2	3b2	变体1	e_3	无效输入
3	2.c	变体1	e_4	无效输入
4	a7.0	变体1	e_6	无效输入
5	+b	变体1	e_7	无效输入
6	001.9b	变体1	e_11	无效输入
7	1t.4	变体1	e_12	无效输入
8	..3	变体2	e_2	无效输入
9	2+3	变体2	e_3	无效输入
10	2..	变体2	e_4	无效输入
11	.4.3	变体2	e_6	无效输入
12	++	变体2	e_7	无效输入
13	5..5	变体2	e_12	无效输入
14	.4	变体3	e_2	无效输入
15	3.	变体3	e_4	无效输入

Offer

续表

测试用例	被输入浮点数	被覆盖的变体	被覆盖的语法元素	测试范围
16	-	变体 3	e_7	无效输入
17	.	变体 3	e_17	无效输入
18	2.s9	变体 4	e_1	无效输入
19	e+2.3	变体 4	e_5	无效输入
20	+t3.4	变体 4	e_5	无效输入
21	+3y.6	变体 4	e_5	无效输入
22	4c5.6	变体 4	e_11	无效输入
23	345v.8	变体 4	e_12	无效输入
24	r0.9	变体 4	e_13	无效输入
25	0g.7	变体 4	e_13	无效输入

3.4 白盒测试及测试用例设计方法

白盒测试是面试官们常考的题目类别之一。面试官通常希望通过考查应试人员的白盒测试水平来了解应试人员的技术功底（通常更靠近编程方面的功底）。确实，测试人员通过对白盒测试知识的学习可以提高对被测代码的理解能力，并且通过白盒测试可以发现很多只运用黑盒测试无法发现的软件缺陷。

3.4.1 白盒测试定义

面试试题 1

什么是白盒测试？

解答

白盒测试也称结构测试或逻辑驱动测试，它是通过分析被测组件内部工作原理，通过测试来检测被测组件内部的运行是否符合产品规格说明书的规定。

通常来说，白盒测试是对应于黑盒测试来定义的。众所周知，黑盒测试就是测试人员把软件产品（或阶段性产品）看做是一个黑盒。在测试过程中测试人员只需关心对这个软件黑盒进行操作会得到什么样的结果，而不必深入地去了解它的内部实现机制所进行的测试活动。这个软件黑盒可以是一个产品，也可以是一个组件，甚至可以是一个函数。测试人员通过组件暴露的公共接口分析它的输入域和输出域，然后根据其分析结果编写测试数据。

对应于黑盒测试，白盒测试要求测试人员打开软件黑盒去了解开发人员的代码实现细节，这些细节包括数据流和控制流。

Offer!

- 数据流方面：进出组件的数据是否能被正确地处理、组件中用于计算使用的数据或作为谓词使用的数据是否被正确使用、是否有冗余、其数据类型是否运用得当。
- 控制流方面：程序中的每一条代码是否都有意义、程序中是否有无法被执行到的语句、程序中的判定是否正确、程序中的各条路径是否正确。

白盒测试通常用于单元测试阶段。单元测试一般处于产品开发的较早阶段，一般由开发人员独立完成或由开发人员与测试人员合作完成，在这个阶段进行充分的白盒测试可以极大地降低软件的开发成本。

我们可以试想，如果一个在白盒测试阶段应当被发现并更正的软件缺陷被遗漏到系统测试阶段甚至用户验收测试阶段，那么首先，当产品集成之后，各组件之间的关系变得越来越复杂，定位这个缺陷就会变得越来越困难。其次随着项目的向前推进，越来越多的角色会被卷入到对缺陷的讨论，这无疑加大了沟通的成本。在开发阶段修改一个软件缺陷只需要对其进行单元测试级别的回归测试来验证其是否被更正，而在后期所有缺陷在修改之后必须被统一地审查以评估新的修改对现阶段的产品不会产生负面的影响，通过之后才能进入正式发布的测试版本中。之后还必须经过一个完整的验证版本有效性的测试阶段（Sanity Test 或 Build Verification Test）才会进入回归测试阶段，以验证缺陷是否真的被更正。这无疑大大地加长了一个缺陷从被发现到被更正的周期。

通常来说，一个缺陷在产品上市之后被发现和在单元测试阶段被发现其更正的成本相差可以高于 100 倍。

面试试题 2

白盒测试与黑盒测试有什么区别？

解答

黑盒测试。

完全不考虑程序内部结构和内部特性。

检验程序是否能适当地接收输入数据而产生正确的输出信息。

多用于针对软件界面、软件功能、性能、安全性等方面进行测试。

“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用。

适用于软件测试的各个阶段（单元测试、组件测试、集成测试、系统测试及用户验收测试）。

白盒测试。

全面了解程序内部逻辑结构、对所有逻辑路径进行测试。

对程序的数据流和所有分支、路径进行覆盖。

多用于对软件内部实现机制的正确性、有效性进行检查。

“白盒”法是穷举路径测试。

多用于软件测试的单元测试阶段。

Offer!

3.4.2 白盒测试方法

本节内容将由浅入深地介绍白盒测试方法。在逻辑覆盖方面，将按照语句测试、分支/判定测试、条件测试、基本路径测试的顺序来介绍，其覆盖的深度逐步加深。但这些对路径的覆盖是有其局限性的。

- 路径测试不能查出程序违反了设计规范，即程序本身是错误的。
- 路径测试不能查出程序中遗漏路径的错误。
- 路径测试可能发现不了一些与数据相关的错误。

此外还将介绍数据流测试等方法。

1. 语句测试（Statement Testing）

语句测试是一种结构化测试技术，该测试技术要求被测组件能够被分解为若干个构成语句。在该模式下所设计的测试用例应当覆盖所有的可执行语句。测试用例应当包括以下几个部分。

- 对该组件的输入。
- 该测试用例所覆盖的可执行语句。
- 测试用例的期望输出。

基于语句测试模式设计的测试用例可通过以下公式计算覆盖率：

语句测试覆盖率=被执行的可执行语句数/可执行语句总数×100%

在介绍这种技术之前先来解释下面两个问题。

- 什么是语句？
- 怎样的语句是可执行的？

总的来说一条语句必须具有原子性，也就是说在执行一条语句时，该语句所包含的操作要么一次性都执行了，要么都没执行。例如代码 *if a then b* 就不能被看作是一条语句，因为 b 执行与否依赖于 a 的执行结果。

在语句测试模式中，语句的定义并不需要与组件开发语言中的语句的定义一致。在此，语句的可执行性通常表现为与之对等的机器码的可执行性。因此我们将以下操作都看作是可执行的。

- 赋值。
- 循环和选择。
- 过程或函数调用。
- 通过显式初始化所声明的变量。
- 堆中动态分配的变量存储空间。

然而，大多数其他的变量声明可被看作是非可执行的，因为它们可能并不分配内存或执行什么操作。

基于该模式开发的测试用例的目的就是覆盖所有的可执行语句。让我们来看以下的 C 语言代码：

```
a;
```



```
if (b)
{
c;
}
d;
```

我们可以看到当 b 为 true 时，所有的可执行语句 a、b、c、d 都将被覆盖，因而达到了 100%的可执行语句的覆盖率。然而当 b 为 false 时，c 语句不能被覆盖，此时就达不到 100%的可执行语句的覆盖率。

2. 分支/判定测试（Branch/Decision Testing）

分支/判定测试基于以下模型。在该模型中被测组件包含有若干个判定。组件被执行时经过每一个判定后，根据判定的结果不同会产生多个包含可执行语句的分支。通常来说，循环和选择语句中都会包含判定。

在该模式下设计的测试用例应当覆盖所有判定的分支。测试用例应包括以下部分。

- 对被测组件的输入。
- 被该测试用例所覆盖的判定和分支。
- 该测试用例的期望结果。

在该模式下设计的测试用例用于覆盖程序中的每个判定所产生的“真值”或“假值”，或者说使得程序中的每一个取“真”分支和取“假”分支至少经历一次（如果是 switch case 语句则判定的结果可能出现多个分支）。

分支测试可通过以下公式计算覆盖率：

分支覆盖率=被执行的分支数/分支的总数×100%

判定测试可通过以下公式计算覆盖率：

判定覆盖率=被执行的判定所产生的分支数/判定所产生的分支的总数×100%

例子

该组件是一个函数，该函数利用辗转相除法求两个大于 0 的正整数的最大公约数。

该组件的源代码如下：

```
int greatestCommonDivisor ( int a,  int b )
{
  if  ((a>0) && (b>0)) {
        int temp = 0;
        while  (b) {
            temp=a%b;
            a=b;
            b=temp;
        }
        return a;
   }
      else return -1;
```

Offer

```
}
```

下面来介绍一下如何在分支/判定测试模式下设计测试用例。首先需要对被测组件进行分析，并画出其控制流图。

说明

程序的控制流图：描述程序控制流的一种图示方法。控制流图只有两种图形符号：

- 图中的圆称为流图的结点，表示一条或多条无分支的语句。
- 图中的箭头称为边或连接，代表控制流。

控制流图中由边和结点限定的范围称为区域。

下面先对上例中的被测组件的源码进行区域划分。划分的原则是每一个被划分出的区域中的一条或多条语句要么都被执行，要么都不会被执行，也就是说每个区域中不会有多余一个分支。因此，上例中的被测源码可划分如下：

```
int greatestCommonDivisor ( int a,  int b )
{                                                                    //B1
  if  ((a>0) && (b>0)) {                                             //B2
        int temp = 0;                                                //B3
        while  (b) {//check if b==0                                  //B4
              temp=a%b;
              a=b;
              b=temp;                                                //B5
        }                                                            //B6
        return a;
      }                                                              //B7
      else return -1;                                                //B8
}
```

下面把被划分出来的区域作为控制流图的节点，根据控制流的走向用箭头连接起来。于是，上例中的被测组件可对应于以下控制流图，如图 3-12 所示。

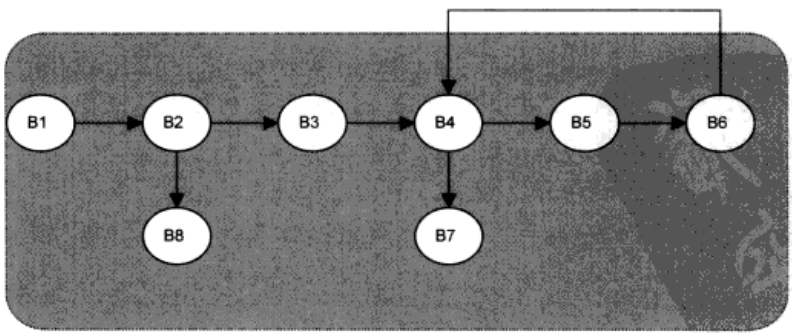


图 3-12　被测组件对应的控制流图

由上图可以轻易看出，区域 B2 和 B4 是判定。因为其有多余一个的出口（都产生两个分支，所以两个判定一共产生了 4 个分支）。而分支的总数为 8，因为图中有 8 个箭头。

下面的任务就是要设计测试用例来覆盖所有的判定和分支，因此我们设计以下用例来覆盖所有的判定和分支，如表3-21所示。

表3-21 设计以下用例来覆盖所有的判定和分支

测试用例	输入		所覆盖的判定和分支	期望输出	分支覆盖率	判定覆盖率
	a	b				
001	0	1	B1->B2->B8	-1	25%	25%
002	36	24	B1->B2->B3->B4->B5->B6->B4->B7	12	87.50%	75%
合计	NA	NA	所有判定和分支	NA	100%	100%

现在来分析一下两条测试用例的分支覆盖率和判定覆盖率。根据前面的分析我们知道该被测组件共有两个判定，一共产生了4个分支，而分支的总数为8，因为其控制流图中有8个箭头。测试用例001覆盖了两个分支和一个判定（B2）所产生的1个分支。因此其分支覆盖率为2/8×100%=25%，其判定覆盖率为1/4×100%=25%。测试用例002覆盖了7个分支和2个判定（B2，B4）所产生的3个分支，因此其分支覆盖率为7/8×100%=87.50%，其判定覆盖率为3/4×100%=75%。合计，两个测试用例覆盖了所有的判定和分支。

3. 条件测试（Condition Testing）

条件测试是一种结构化测试方法。它包括3种测试方法，分别为：条件分支测试、条件分支组合测试和修正条件判定测试。这3种测试方法的原理都非常相似。下面用一个例子来阐述这3种测试方法如何使用。

让我们来看一段代码：

```
if （A||（B&&C））
{
    Some codes here…
}
else
{
    Some codes here…
}
```

以上代码中A、B、C为3个布尔操作数。其操作数本身可以是布尔常量（true， false），也可以是一组表达式计算出来的布尔值（例如：m>n）。

下面将对被测代码进行条件分支测试。

条件分支测试要求一个判定中的每一个布尔操作数的真值和假值都被测试用例覆盖（执行）至少一次。根据上述要求只要对上例中的布尔操作数A、B、C的真值和假值都覆盖一遍即可。

每一个利用此方法设计的测试用例都应当包括。

- 对被测组件的输入。
- 被测判定中各布尔操作数的取值及判定的结果。

Offer

- 期望的输出。

于是可以设计以下用例，如表 3-22 所示。

表 3-22 设计以下用例（一）

测试用例	布尔操作数			判定结果
	A	B	C	
1	TRUE	TRUE	TRUE	TRUE
2	FALSE	FALSE	FALSE	FALSE

这样也同样做到了对被测代码的条件分支覆盖。

当然，我们也可以把用例设计为，如表 3-23 所示。

表 3-23 设计以下用例（二）

测试用例	布尔操作数			判定结果
	A	B	C	
1	TRUE	FALSE	FALSE	TRUE
2	FALSE	TRUE	TRUE	TRUE

这样被测代码的条件分支就没有被完全覆盖覆盖。

由此可以看出，条件分支测试是一种很弱的测试覆盖方法，它甚至无法确定是否一定都覆盖了一个判定的真值结果分支和假值结果分支（如表 3-23 中的两个测试用例）。

下面将对被测代码进行条件分支组合测试。

条件分支组合测试要求一个判定中的所有布尔操作数的所有取值组合都被测试用例覆盖（执行）至少一次。根据上述要求需要对上例中的布尔操作数 A、B、C 各取真值和假值，然后对其进行组合，因此将会产生 $2\times2\times2=8$（即 2^n 个测试用例，其中 n 为布尔操作数的个数）。

每一个利用此方法设计的测试用例应当包括：

- 对被测组件的输入。
- 被测判定中各布尔操作数的取值及判定的结果。
- 期望的输出。

于是可以设计以下用例，如表 3-24 所示。

表 3-24 设计以下用例（三）

测试用例	布尔操作数			判定结果
	A	B	C	
1	TRUE	TRUE	TRUE	TRUE
2	FALSE	TRUE	TRUE	TRUE
3	TRUE	FALSE	TRUE	TRUE
4	FALSE	FALSE	TRUE	FALSE
5	TRUE	TRUE	FALSE	TRUE

Offer!

续表

测试用例	布尔操作数			判定结果
	A	B	C	
6	FALSE	TRUE	FALSE	FALSE
7	TRUE	FALSE	FALSE	TRUE
8	FALSE	FALSE	FALSE	FALSE

下面将对被测代码进行修正条件判定测试。

修正条件判定测试是在条件分支组合测试的基础上发展而来的，它对其条件分支组合测试用例集合进行合理优化。修正条件判定测试被广泛应用于航空软件的开发测试（RTCA/DO-178B标准）。

修正条件判定测试要求一个判定中的某一个布尔操作数在其他布尔操作数取值不变的情况下，当其取真值或假值时，能够独立地影响该判定的结果。根据上述要求我们要对上例中的布尔操作数A、B、C逐个取真值或假值以产生测试用例。并且对于同一个布尔操作数，每一次取值都会使该判定产生不同的结果。因此我们总共最多将会产生6个测试用例（即2×n个测试用例，其中n为布尔操作数的个数）。

每一个利用此方法设计的测试用例应当包括。

- 对被测组件的输入。
- 被测判定中各布尔操作数的取值及判定的结果。
- 期望的输出。

对布尔操作数A设计以下用例（B、C的取值不变），如表3-25所示。

表3-25 对布尔操作数A设计以下用例

测试用例	布尔操作数			判定结果
	A	B	C	
A1	TRUE	FALSE	TRUE	TRUE
A2	FALSE	FALSE	TRUE	FALSE

对布尔操作数B设计以下用例（A、C的取值不变），如表3-26所示。

表3-26 对布尔操作数B设计以下用例（一）

测试用例	布尔操作数			判定结果
	A	B	C	
B1	FALSE	TRUE	TRUE	TRUE
B2	FALSE	FALSE	TRUE	FALSE

对布尔操作数B设计以下用例（A、B的取值不变），如表3-27所示。

Offer!

表 3-27 对布尔操作数 B 设计以下用例（二）

测试用例	布尔操作数			判定结果
	A	B	C	
C1	FALSE	TRUE	TRUE	TRUE
C2	FALSE	TRUE	FALSE	FALSE

我们不难发现在上述 3 张测试用例列表中，A2 与 B2， B1 与 C1 布尔操作数的组合是相同的。因此上述 3 张表可以整合为下表，如表 3-28 所示。

表 3-28 上述 3 张表可以整合如下

测试用例	布尔操作数			判定结果
	A	B	C	
A1	TRUE	FALSE	TRUE	TRUE
A2，B2	FALSE	FALSE	TRUE	FALSE
B1，C1	FALSE	TRUE	TRUE	TRUE
C2	FALSE	TRUE	FALSE	FALSE

实际上，通过使用修正条件判定测试用例设计方法，最多需要 2×n 个测试用例，最少需要 n+1 个测试用例就可以做到 100%修正条件判定覆盖。

说明

布尔表达式的优化：

有一些编程语言对布尔操作符进行了短路处理（即所谓的优化）。例如在 C/C++中，如果布尔与（&&）操作符的第一个操作数为 False，则将不再对第二个操作数进行判断（如果是表达式则不进行计算）。如果布尔或（||）操作符的第一个操作数为 True，则将不再对第二个操作数进行判断（如果是表达式则不进行计算）。

这样的短路设计对条件分支测试和修正条件判定测试没有什么影响，但是对于条件分支组合测试来说，则不可能达到 100%的覆盖率。

到现在为止，本节讨论的测试技术都是基于通过布尔条件来控制判定的实例，而实际上在编程语言中还有其他种类的判定。例如有多个选择分支的判定：switch-case 语句、goto 语句；又如循环语句：for、while。对于这些种类的判定一样可以将其等价于上例中通过布尔条件来控制的判定，然后设计测试用例。

4．数据流测试（Data Flow Testing）

数据流测试是一种结构化测试方法。其目的是为了覆盖被测组件中所有变量从被定义点到被使用点的路径，该路径被称为定义使用对路径（definition-use pairs，简称 du-pairs）。

我们沿用以前的一个例子来说明该项方法的使用。

该组件是一个函数，该函数利用辗转相除法求两个大于 0 的正整数的最大公约数。该组件

Offer

的源代码如下：

```
int greatestCommonDivisor ( int a,  int b )                    --0
{                                                              --1
  if  ((a>0) && (b>0)) {                                       --2
        int temp = 0;                                          --3
        while  (b) {                                           --4
             temp=a%b;                                         --5
             a=b;                                              --6
             b=temp;                                           --7
        }                                                      --8
        return a;                                              --9
   }                                                           --10
      else return -1;                                          --11
}                                                              --12
```

在对其设计用例之前先介绍以下几个概念：

- 变量的定义点：表示某个变量在代码的某一行定义，此时的定义包括把值输入给变量、为变量赋值以及过程调用等。变量的值往往会随着这些语句的执行发生变化。
- 变量的使用点：表示某个变量在代码的某一行被使用过。使用包括将变量的值输出语句、将变量的值赋值给其他变量、循环语句、条件语句、过程调用语句都是变量的使用点。变量的值不会随着这些语句的执行发生变化。

变量的使用又包括以下两种。

（1）P（predicate）-use 即谓词使用，例如 a>0。谓词使用语句执行之后程序的出口数≥2。

（2）C（computation）-use 即计算使用，例如 a%b。计算使用语句执行之后程序的出口数≤1。

下面列出被测组件中所用到的变量：a、b、temp。

然后列出这 3 个变量在被测组件中的定义和使用情况，如表 3-29 所示。

表 3-29　　3 个变量在被测组件中的定义和使用情况

行　号	定　义	谓词使用（p-use）	计算使用（c-use）
0	a，b		
2		a，b	
3	temp		
4		b	
5	temp		a，b
6	a		b
7	b		temp
9			a

下一步来找出各个变量基于谓词使用或计算使用的定义使用对，如表 3-30 所示。

Offer

表 3-30 各个变量基于谓词使用或计算使用的定义使用对

定义使用对路径	谓词使用（p-use）	计算使用（c-use）
0->2	a，b	
0->4	b	
0->5		a，b
0->6		b
3->7		temp
5->7		temp
6->9		a
7->4	b	
7->5		b
7->6		b
6->5		a

下面对其设计测试用例。

（1）全定义覆盖（All-definition）。

为了取得 100%的定义覆盖，我们至少需要覆盖每一个变量从其所有定义到某一个使用的定义使用对（definition-use pair）路径（可以是谓词使用，也可以是计算使用），如表 3-31 所示。

表 3-31 全定义覆盖

测试用例	全定义			输入		输出
	变量	定义使用对	路径	a	b	
1	a，b	0->2	0-2	-1	2	-1
2	temp	3->7	3-7	12	6	6
3	temp	5->7	5-7	12	6	6
4	a	6->9	6-7-8-4-9	12	6	6
5	b	7->4	7-8-4	12	16	4

（2）全谓词使用覆盖（All-p-uses）。

为了取得 100%的全谓词使用覆盖，我们至少需要覆盖每一个变量从定义到其所有谓词使用的某一个定义使用对路径，如表 3-32 所示。

表 3-32 全谓词使用覆盖

测试用例	全谓词使用			输入		输出
	变量	定义使用对	路径	a	b	
1	a，b	0->2	0-2	-1	2	-1
2	b	0->4	0-4	12	6	6
3	b	7->4	7-8-4	12	16	4

（3）全计算使用覆盖（All-c-uses）。

为了取得 100%的全计算使用覆盖，我们至少需要覆盖每一个变量从定义到其所有计算使用

的某一个定义使用对路径，如表3-33所示。

表3-33 全计算使用覆盖

测试用例	全计算使用			输入		输出
	变量	定义使用对	路径	a	b	
1	a，b	0->5	0-5	12	6	6
2	b	0->6	0-6	12	6	6
3	temp	3->7	3-7	12	6	6
4	temp	5->7	5-7	12	6	6
5	a	6->9	6-7-8-4-9	12	6	6
6	b	7->5	7-8-4-5	12	16	4
7	b	7->6	7-8-4-5-6	12	16	4
8	a	6->5	6-7-8-4-5	12	16	4

（4）全使用覆盖（All-uses）。

为了取得100%的使用覆盖，我们至少需要覆盖每一个变量从定义到其所有使用的某一个定义使用对路径。将全谓词使用覆盖和全计算使用覆盖的测试用例合起来就可以得到全使用覆盖的测试用例，如表3-34所示。

表3-34 全使用覆盖

测试用例	全使用			输入		输出
	变量	定义使用对	路径	a	b	
1	a，b	0->5	0-5	12	6	6
2	b	0->6	0-6	12	6	6
3	temp	3->7	3-7	12	6	6
4	temp	5->7	5-7	12	6	6
5	a	6->9	6-7-8-4-9	12	6	6
6	b	7->5	7-8-4-5	12	16	4
7	b	7->6	7-8-4-5-6	12	16	4
8	a	6->5	6-7-8-4-5	12	16	4
9	a，b	0->2	0-2	-1	2	-1
10	b	0->4	0-4	12	6	6
11	b	7->4	2007-8-4	12	16	4

（5）全定义使用对路径覆盖（All-du-path）。

为了取得100%的定义使用对路径覆盖，我们至少需要覆盖每一个变量从定义到其所有使用的定义使用对的所有路径。该覆盖方法与全使用覆盖的区别在于全使用覆盖只需覆盖一个定义使用对的某一个路径，而全定义使用对路径覆盖必须覆盖一个定义使用对的所有可能路径。关于其测试用例集合在此就不再一一列举了，有兴趣的读者可以自行尝试列举。

Offer

5. 基本路径测试

基本路径测试法是利用控制流图对被测组件的运行路径进行分析，然后推导出被测组件的基本路径，最后开发测试用例来覆盖这些路径。基本路径测试法要求被测组件的所有语句至少被测试用例集覆盖一次。

在该模式下设计的测试用例应当覆盖所有路径。测试用例应包括以下部分。

- 对被测组件的输入。
- 被该测试用例所覆盖的路径。
- 该测试用例的期望结果。

基本测试可通过以下公式计算覆盖率：

基本路径覆盖率=被执行的路径数/路径的总数×100%

我们依然使用之前使用过的一个例子——某组件是个函数，功能是利用辗转相除法求两个大于 0 的正整数的最大公约数。该组件的源代码如下：

```
int greatestCommonDivisor ( int a,  int b )
{                                                                    //B1
  if  ((a>0) && (b>0)) {                                             //B2
        int temp = 0;                                                //B3
        while  (b) {//check if b==0                                  //B4
              temp=a%b;
              a=b;
              b=temp;                                                //B5
        }                                                            //B6
        return a;
      }                                                              //B7
      else return -1;                                                //B8
}                                                                    // B9
```

下面绘出其控制流图。在该设计模式下的控制流图应当注意以下几点。

- 在选择语句等分支结构中，分支结合点处应有一个结点。
- 边和结点形成的环所圈定的区域称为区域。环以外的区域也应记为一个区域。
- 如果判定中的条件表达式是由多个逻辑运算符连接的复合条件表达式，则需要在控制流图将其转换为只有单一条件的嵌套的判定。

因此可以得到以下控制流图，如图 3-13 所示。

在图 3-13 中，把复合条件表达式（（a>0）&&（b>0））转化为了 B2a 和 B2b 两个节点。

下面计算圈复杂度。圈复杂度是一种描述程序逻辑复杂性的软件度量，其对应于独立路径数。

有以下方法计算圈复杂度。

- 给定流图 G 的圈复杂度 V（G）=A， A 是流图中的区域数量。
- 给定流图 G 的圈复杂度 V（G）=E-N+2，E 是流图中边的数量，N 是流图中结点的数量。
- 给定流图 G 的圈复杂度 V（G）=P+1，P 是流图 G 中判定结点的数量。

Offer

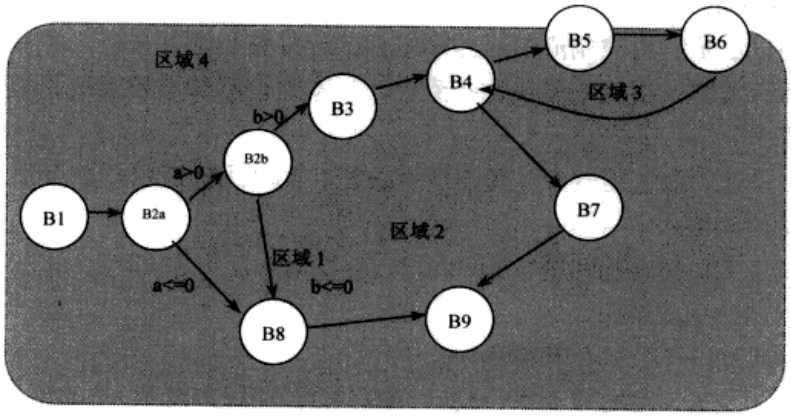


图 3-13 控制流图

该例的流图中有 10 个节点、12 条边、4 个区域，因此有。

- 圈复杂度 V（G）=A=4 或。
- 圈复杂度 V（G）= E-N+2=12-10+2=4 或。
- 圈复杂度 V（G）=P+1=4。

因此该例中有 4 条独立路径（每一次的判定都会产生一条新的路径）。

- 路径 1：B1-B2a-B2b-B3-B4-B5-B6-B7-B4-B9。
- 路径 2：B1-B2a-B2b-B3-B4-B7。
- 路径 3：B1-B2a-B2b-B8-B9。
- 路径 4：B1-B2a-B8-B9。

于是就可以设计测试用例来覆盖这些独立路径了。

6. 线形代码序列与跳转测试（LCSAJ）

一组 LCSAJ（Linear Code Sequence And Jump）被定义为一组顺序执行的代码序列。一组 LCSAJ 的起点可以是一个程序的起点或者是控制流跳转后的入口点，或控制流可跳达的点。其结束点可以是能导致控制流跳转的点或是程序的出口。

LCSAJ 设计模式下的测试用例应当包括。

- 对组件的输入。
- 用例要覆盖的 LCSAJ（指在特定输入下程序如何顺序运行或跳转）。
- 组件的输出。

LCSAJ 设计模式下的覆盖率：

LCSAJ 覆盖率 =被执行的 LCSAJ 数/总的 LCSAJ 数×100%

下面我们依然使用之前使用过的一个例子——某个组件是函数，功能是利用辗转相除法求两个大于 0 的正整数的最大公约数。该组件的源代码如下：

```
int greatestCommonDivisor ( int a,  int b )                    //1
{                                                              //2
```

Offer

```
if ((a>0) && (b>0)) {                                    //3
        int temp = 0;                                    //4
        while (b) {                                      //5
            temp=a%b;                                    //6
            a=b;                                         //7
            b=temp;                                      //8
        }                                                //9
        return a;                                        //10
    }                                                    //11
    else return -1;                                      //12
}                                                        //13
```

下面先分析代码以产生 LCSAJ。以下列出该组件的分支：

（3->4）：须满足（a>0）&&（b>0）

（3->12）：须满足 a<=0 或 b<=0　　此处跳转

（5->6）：须满足 b 不等于零

（5->10）：须满足 b 等于零　　此处跳转

（9->5）：循环　　此处跳转

下面寻找 LCSAJ 的起始点。这个函数从第 2 行开始，然后运行到第 3、5、9 行都可能会跳转到其他行而非顺序运行。

那么第一个 LCSAJ 就起始于第 2 行，程序从第一行开始运行，运行到第 3 行时可能会发生跳转（3->12），因此得到一个 LCSAJ，即（2，3，12）。程序如果未在 3 行发生跳转则运行至第 5 行可能会发生跳转（5->10），因此可得到下一个 LCSAJ，即（2，5，10）。同理还可以第 2 行为起始点的其他的 LCSAJ，（2，9，5）。因此得到以第 2 行为起始点的 LCSAJ 为（2，3，12），（2，5，10），（2，9，5）。

从该组件的分支表中可以看出，该函数在第 3、5、9 行都会发生跳转，因此下一个 LCSAJ 就起始于第 3 行，即得到以第 3 行为起始点的 LCSAJ 为（3，3，12），（3，5，10），（3，9，5）。

同理还可以列出以其他跳转点为起点的 LCSAJ：以第 5 行为起始点：（5，5，10），（5，9，5）；以第 9 行为起始点：（9，9，5）。

此外，还有另外两个跳转点，第 10 行和第 12 行。这两个特殊的跳转点都会“跳转”到该函数的出口第 13 行，因此有另外两个 LCSAJ，即（10，10，13），（12，12，13）。

下面列举所有的 LCSAJ 和用于覆盖这些 LCSAJ 的测试用例，如表 3-35 所示。

表 3-35　　所有的 LCSAJ 和用于覆盖这些 LCSAJ 的测试用例

LCSAJ			需满足的要求	测试用例的输入
起 始 点	结 束 点	跳 转 至		
2	3	12	a<=0 或 b<=0	a=0，　b=-1
2	5	10	无法覆盖	N/A
2	9	5	a>0 并且 b>0	a=5，　b=15

Offer!

续表

LCSAJ			需满足的要求	测试用例的输入
起始点	结束点	跳转至		
3	3	12	a<=0 或 b<=0	a=0， b=-1
3	5	10	无法覆盖	N/A
3	9	5	a>0 并且 b>0	a=5， b=15
5	5	10	经过计算 b=0 且 a>0	a=5， b=15
5	9	5	a>0 并且 b>0	a=5， b=15
9	9	5	经过计算 b>=0 且 a>0	a=5， b=15
10	10	13	经过计算 b=0 且 a>0	a=5， b=15
12	12	13	a<=0 或 b<=0	a=0， b=-1

通过表 3-34 可以看出，有的 LCSAJ 是无法覆盖的。此例中只用了两个测试用例就覆盖了所有可测的 LCSAJ，而实际上对于拥有复杂逻辑的组件可能需要大量的测试用例才能覆盖。

3.5 单元测试

单元测试就是对软件的最基本组成单元进行测试，它是软件测试中最低级别的测试。通常来说在单元测试中，软件的被测单元将在与其他组件相隔离的情况下进行测试。

被测试的单元在不同的编程语言中是有不同定义的。例如在 C、Pascal 这些过程化的语言中被测单元一般是指函数或过程；而在 C++、C#或 Java 等面向对象的语言中被测单元既可以是类，也可以是类中的成员函数。

面试试题1

为什么要对程序进行单元测试？

解答

- 保证被测代码有正确的行为，可以验证代码是否与详细设计一致。

在编写代码时，通常在每次实现一个功能或完成一个函数时都会反复调试以确保其可以编译通过，但编译通过只能说明被编译的代码没有语法问题，但仍需要对其语义的正确性进行测试。这样的简单的测试行为其实就是一种单元测试。如果没有这样的单元测试谁也不能保证这段代码的行为一定是正确的。当然对代码进行简单的单元测试可能只能覆盖代码的一部分功能，这就需要使用各种白盒或黑盒的测试方法对其进行系统的单元测试以提高其单元测试覆盖率。

- 窥探软件内部的实现机制，可以发现其他测试阶段难以发现的软件缺陷。

单元测试是软件测试的最初级阶段，通常在这个阶段的软件产品还处于未集成的阶段。在

Offer

该阶段对产品的基本组成单元的内部原理和对外接口进行详尽地测试检查可以发现许多在其他测试阶段难以发现的软件缺陷。因为在该阶段使用的测试方法大多是白盒测试方法，而在其他测试阶段测试人员很少有机会去使用白盒测试方法去窥探软件内部的实现机制。

- 充分的单元测试可以极大地降低软件的开发成本。

单元测试一般处于产品开发的较早阶段，一般由开发人员独立完成或由开发人员与测试人员合作完成。在该阶段进行充分的单元测试可以极大地降低软件的开发成本。如果一个在单元测试阶段应当被发现的软件缺陷被遗漏到系统测试阶段甚至用户验收测试阶段，那么随着软件产品复杂性的提高定位这个缺陷就会变得越来越困难。随着项目的向前推进，越来越多的角色会被卷入到对缺陷的讨论中，这无疑加大了沟通的成本，而且这样一个缺陷从被发现到被更正的周期也会被大大延长。

面试试题2

由谁来做单元测试？

解答

- 这是一道比较有意思的题目。通常来说在应聘软件测试工程师的面试过程中有些面试官会问应试人员是否做过单元测试或白盒测试（虽然单元测试和白盒测试不能完全打等号）。对于绝大多数应试人员来说，如果一直从事测试工作而非编写代码工作，那么接触单元测试或白盒测试的机会还是很少的。
- 因此不难看出单元测试与其他测试阶段的不同。单元测试通常是由软件开发人员来完成的，因为单元测试阶段通常被看作是编码工作的一部分。也就是说，经过了充分的单元测试的代码才算是已完成的代码，才可以被提交到代码库中，而且在对被测代码进行的单元测试活动中所产生的所有文档、测试用例、工具代码及测试结果等都要一一存档以备测试部门的审核，当然也不排除一些公司有专门的单元测试团队，其组成人员一般都会有一定的开发背景，或者其团队由具有一定的开发背景的工程师与专门的测试工程师组成。

面试试题3

什么时候开始做单元测试？

解答

- 单元测试活动应当在软件的编码阶段进行，该阶段一般位于软件开发团队已经给出了详细的软件设计文档并通过了相关人员的审核之后。
- 在编码阶段，开发人员一般先设计好各单元之间的接口（或类的接口、函数的接口等），然后再实现单元的内部代码，最后就可以对各个单元进行测试。

Offer!

说明

当然，在有些开发理论中（测试驱动开发-Test Driven Development），要求先编写测试代码，再进行软件开发。

面试试题4

怎么做单元测试？

解答

- 我们知道，在编码阶段，开发人员一般先设计好各单元之间的接口（或类的接口、函数的接口等），然后再实现单元的内部代码，最后就可以对各个单元进行测试。

因此对于每个被测单元，应首先知道其对外部暴露的接口（全局变量、类的公有成员、函数的形参、返回值、函数名等）。我们通过组件暴露的公共接口分析它的输入值域，然后根据其输入值域编写测试数据。在这里经常用到的是黑盒测试用例设计方法所设计的测试用例，可以使用的黑盒测试方法有等价类划分、边界值分析、语法测试等方法。

之后可以对被测单元的内部进行白盒测试。白盒测试要求测试人员打开软件黑盒去了解开发人员的代码实现细节，这些细节包括数据流和控制流。可以使用的白盒测试方法有语句测试、分支/判定测试、条件测试、基本路径测试及线形代码序列与跳转测试及数据流测试等。

- 测试被测单元时，由于单元的独立性，通常需要编写驱动函数或桩函数来模拟被测单元的被调用和调用。

说明

驱动模块：在测试被测模块时，一般需要在一个模块中构造一些实验数据作为传入被测模块的输入，然后再在所构造的模块中调用被测模块。这样的模块被称为驱动模块（一般为main函数）。

桩模块：在测试被测模块时，被测模块可能要调用一个未被完成的模块（或一连串模块的调用）。这时测试人员可以写一个临时的、简单的模块以模拟被调模块的部分或全部行为。这样的模块被称为桩模块。

- 单元测试的测试策略。

（1）自顶向下。

在此测试模式中通常先测最上层的单元。待上层的被测单元测试完成之后，再用上层的已测单元作为下层被测单元的驱动模块，进而测试下层模块，依此类推。

自顶向下模式的优点。

- 测试人员不用编写驱动模块。

自顶向下模式的缺点。

- ◆ 当底层的模块未开发完成时，测试人员必须自行编写桩模块。

Offer

- 当测试到底层模块时，如果发现缺陷或底层模块的接口设计有问题，则势必会影响到上层的已测单元，这样，上层的已测单元也必须做出相应的修改。
- 下层单元必须等到上层单元开发并测试完成后才能进行测试。
- 使用上层已测单元作为驱动模块，不能保证对下层被测单元的覆盖率，因为上层已测单元并不是专门为单元测试设计的测试代码。

（2）自底向上。

在此测试模式中通常先编写驱动模块测试最底层的单元。待底层的被测单元测试完成之后，再用底层的已测单元作为上层被测单元的桩模块，进而测试上层模块，依此类推。

自底向上模式的优点。

- 测试人员不用编写桩模块。
- 后期发现的上层被测单元的缺陷一般不会对底层产生影响。

自底向上模式的缺点。

- 测试人员必须自行编写驱动模块。
- 上层单元必须等到下层单元开发并测试完成后才能进行测试。
- 使用底层已测单元作为桩模块，不能保证对上层被测单元的覆盖率，因为底层已测单元并不是专门为单元测试设计的测试代码。

（3）独立测试。

在此测试模式中通常为每个被测单元编写驱动模块和桩模块，独立地对被测单元进行测试。这是在实践中被广泛应用的一种模式。

独立测试模式的优点。

- 驱动模块和桩模块的覆盖率可以得到保证。
- 对不同层次上的单元可以并行地进行测试。

独立测试模式的缺点。

- 测试人员必须自行编写驱动模块和桩模块。

面试试题5

我们知道，在编码阶段，开发人员一般先设计好各单元之间的接口（或类的接口），此时函数单元测试的入口条件（Entry Criteria）和出口条件（Exit Criteria）是什么？

解答

（1）入口条件。

在做单元测试之前都需要做一些准备工作，包括。

- 被测单元编码已完成。
- 代码编译通过。
- 通过了静态代码检查工具的检测。



- 至少完成一轮代码检视或走读，并通过。
- 测试用例已被检视通过。
- 工具代码已完成并被检视通过。

（2）出口条件。

单元测试完成后如果完成以下条件，则下一阶段的测试就可以进行。

- 所有的测试用例都被执行。
- 测试覆盖率达到了预定的要求。
- 被测模块的缺陷率低于预定的要求。

3.6 集成测试

集成测试就是对软件已测试完成的单元或组件之间的接口进行测试，其又称为组装测试或子系统测试。

在最简单的情况下，当两个已完成单元测试的单元要组成一个组件时，就必须对这个组件进行集成测试。测试通过后将更多的组件聚合进行更大范围的集成测试从而得到更大的组件，进而组成模块、子系统和系统。

因此在较大规模的开发项目中，集成测试通常按照以下层次进行。

- 组件内部各单元之间的集成，主要对已完成单元测试之后基本单元之间接口进行测试。
- 模块内部的集成，主要测试的是各个组件之间的接口。
- 子系统中各功能模块之间的集成，主要测试子系统中各个模块的交互。
- 系统中各子系统之间的集成，主要测试系统中各子系统之间、子系统与模块之间的交互。

集成测试需要窥探程序内部的接口，但有大量使用的是黑盒测试的方法，所以它结合了白盒测试和黑盒测试的特点，常被归类为“灰盒测试”。

面试试题1

为什么要对程序进行集成测试？

解答

- 可以发现在单元测试阶段难以发现的缺陷

对软件基本单元进行单元测试的目的是为了检验该被测单元的正确性与否。这与集成测试的目的有本质上的区别。有时基本单元单元的功能虽然正确实现了，但其给出的接口并不正确。类似这样的缺陷通常很容易在集成测试阶段被发现。

- 结合白盒测试与黑盒测试的特点对被测组件进行测试

软件在集成测试阶段，测试人员还有机会对被测组件的内部实现机制进行测试，有时又有

Offer!

机会从更加宏观的角度运用黑盒测试方法对其进行测试。这是在其他测试阶段难以实现的。

- 逐步地对软件进行集成测试可以有效地定位软件因集成新的组件所引入的缺陷

一般来说软件的集成是逐步进行的，并且各个组件之间一般都有依赖关系。如果在测试软件产品时，当完成了对所有基本单元的单元测试之后就立即将所有的基本单元聚合到一起进行系统测试，那么当测试人员发现模块间交互问题的缺陷时就很难界定这个缺陷属于哪一个模块。因为我们根本不知道哪一个模块是稳定的，其模块的内部也没有进行过集成测试。

当在稳定的模块或系统中加入新的模块并出现问题时，我们比较容易判断该问题出现在与新模块有交互模块的接口上。而这些模块的内部都是稳定的，此时只要对少数提供这些接口的函数进行排查即可。

- 逐步地对软件进行集成测试可以有效地评估被集成的新模块对原系统的影响

只有对软件产品进行逐步集成测试，才能形成稳定的、已完成集成的模块或系统。当在该稳定的模块或系统中加入新的模块时并出现问题时，我们可以评估该组件的引入对原系统的影响，并且结合产品开发的日程安排来决定是否保留该组件。

⚠ 警示

在日程安排紧张的情况下，如果我们将未经过任何集成测试的模块放在一起，则当出现问题时，我们将无法知道将某一个模块摒弃是否能使剩下的模块稳定运行。

- 充分的集成测试可以降低软件的开发成本

集成测试处于产品开发的较早阶段。软件的复杂度要小于此后的测试阶段，定位缺陷时间也会短于其他阶段。一个集成测试阶段缺陷从被发现到被更正的周期也会短于此后的测试阶段。

面试试题2

集成测试是为了验证哪个文档？

解答

在软件的开发过程中，开发人员通常需要将软件的需求文档转化为软件的概要设计文档。在该文档中通常会指明各个模块之间的接口。有的开发团队甚至会专门开发一个软件接口说明（Software Interface Specification）文档以说明各基本单元或模块之间的接口关系。这样的接口关系通常可以用一个接口关系图，即一个无环有向图来表示。

面试试题3

什么时候开始做集成测试？

解答

集成测试活动应当在被测组件的编码完成之后进行。这时的被测组件中各个基本单元已经

Offer!

顺利通过了单元测试但还并未被集成为一个稳定的组件。

测试人员可以将被测组件中功能相近的基本单元先进行集成，待稳定之后再将其他单元逐步地集成到已稳定的部件中。

面试试题4

怎么做集成测试？

解答

集成测试的方法有很多。在本节中将介绍自顶向下集成法、自底向上集成法、三明治集成法（从两端向中间集成）以及大爆炸试集成。各种集成方法都有其优点和缺点。

（1）自顶向下集成法。

自顶向下集成法是从程序的最顶层的模块开始集成，通常是从程序的主函数开始集成。顶层的模块自身先做集成，此时顶层模块某些功能通常要调用底层模块函数或类的接口，而这些底层模块函数或类的接口并没有进行过模块间的集成测试，甚至还未进行过单元测试。因此在自顶向下集成法中这些底层模块的函数或类的接口就必须用桩函数（或桩模块）来代替。

自顶向下集成法按所测试的模块的优先级又可以分为按深度优先集成和按广度优先集成。在测试一个子系统或相对完整的功能模块时，按深度优先集成比较容易将该子系统或功能模块测试的更透彻。

优点。

- 从顶层的功能模块开始集成与测试，较早地验证了用户可以直接接触到的主要的功能模块。
- 逐步的进行自顶向下的集成测试，可以比较容易地隔离缺陷和定位缺陷。
- 不需要为被测组件编写驱动函数。

缺点。

- 从顶层的功能模块开始集成与测试，需要为上层的模块编写桩模块。这增加了测试人员很大的工作量。
- 由于底层模块进行集成测试较晚，因而底层会在顶层模块集成测试完成之后的阶段被发现。然而底层模块所发现的缺陷在较大程度上都会引起上层模块的修改。

（2）自底向上集成法。

自底向上集成法是从程序的最底层的模块开始集成。底层的模块自身先做集成，此时顶层模块进行过模块间的集成测试，甚至还未进行过单元测试。因此在自底向上集成法中就必须用驱动函数来代替上层模块对底层模块的调用。

优点。

- 较早地验证了基础模块，底层模块中的缺陷可以更早地被发现。
- 不需要为被测组件编写桩模块。

Offer!

缺点。

- 从底层的功能模块开始集成与测试，需要为底层的模块编写驱动函数。这增加了测试人员很大的工作量。
- 较自顶向下集成法而言，不容易隔离缺陷和定位缺陷。

（3）三明治集成法（从两端向中间集成）。

顾名思义，三明治集成法就是从软件系统的两端向中间集成，这种集成方法兼具了自顶向下和自底向上集成法的优点。除此之外它还可以使得集成测试从两端向中间并行进行，是现实项目中十分常用的方法。

（4）大爆炸式集成法。

大爆炸式集成法就是等到系统的所有模块都完成了单元测试之后，将所有的模块放到一起做集成测试。这种集成方式首先要将集合到一起的所有模块调通，让系统“跑”起来。

优点。

- 系统中的所有模块都完成了单元测试，因此做集成测试时通常不需编写驱动函数或桩模块。
- 因为所有的模块都集中到了一起，所以测试人员可以直接利用该系统的用户接口进行测试。

缺点。

- 如果不编写驱动函数或桩模块对系统进行集成测试，通常只能测试到用户接口可以直接或间接调用的模块，而并不能详尽地测试所有的模块。
- 测试发现的缺陷不易被隔离和定位。

面试试题5

集成测试的入口条件和出口条件是什么？

解答

入口条件（Entry Criteria）

在做集成测试之前都需要做一些准备工作，其包括。

- 被测模块的单元测试测试结果已通过并被检视。
- 集成测试策略已被检视并通过。
- 集成测试测试用例已开发完成并被检视通过。
- 所需的工具代码已完成并被检视通过。

出口条件（Exit Criteria）

集成测试完成后如果完成了以下工作就可以进行下一阶段的测试。

- 所有的集成测试用例都被执行。
- 集成测试测试覆盖率达到了预定的要求。

Offer

- 被测模块的缺陷率低于预定的上限。

3.7 系统测试

什么是系统测试？

系统测试就是在所有模块或子系统都完成了集成测试之后，测试人员对整个软件系统按照软件产品需求规格说明书进行的功能验证测试。

说明

软件测试直到这个阶段所做的工作还是一直在对一系列需求文档进行验证。

- 在单元测试阶段，测试人员主要工作是对软件详细设计文档的验证。
- 在集成测试阶段，测试人员主要工作是对软件接口说明文档进行验证。
- 在系统测试阶段，测试人员主要工作是对产品需求规格说明书进行验证。

按照软件产品需求规格说明书进行的功能验证测试就意味着此时的被测软件应当运行在产品需求规格说明书所要求的真实的运行环境中。

例子

如果被测产品是网站，就必须将该网站的运行环境准备好，其包括 Web 服务器的种类、版本，数据库的种类、版本，操作系统的种类、版本，以及服务器硬件的配置（内存、处理器、硬盘、网卡等）。除此之外还必须考虑到用户端的客户机的配置情况，这通常包括用户所使用的操作系统和浏览器的种类和版本。

例子

而当被测产品是手机软件等嵌入式系统时，到系统测试阶段，被测软件所对应的硬件系统就必须已准备好，并且已经经过一系列的测试，再将被测软件植入到硬件系统中进行系统测试。

实际上硬件系统通常在集成测试阶段就可以用来植入软件进行测试了。硬件系统也会经历一个需求、设计、实施、测试的过程。一般到软件的系统阶段，硬件系统就已经相当稳定了。而在软件测试的早期，开发人员和测试人员通常都会使用一种可以在普通 PC 机上运行的模拟器。

例子

例如：Google 的手机平台 Android 就有其可运行在 PC 上的模拟器。

同时，系统测试阶段还将对产品进行其他方面的专项测试，如性能测试、压力测试、健壮性测试和用户易用性测试等。

Offer

面试试题1

为什么要对软件进行系统测试？

解答

主要是由于以下几点。

- 系统测试与此前测试阶段的测试目的不同。单元测试是为了保证被测单元的代码有正确的行为，以及验证代码是否与详细设计一致，而集成测试则是对软件已测试完成的单元或组件之间的接口进行测试。这些接口属于软件的内部接口，而系统测试中的功能测试通常都是测试软件的用户接口。
- 系统测试的测试范围更加广泛。系统测试阶段包括一些专项测试，如性能测试、压力测试、健壮性测试和用户易用性测试等，这些专项测试都必须等到系统集成完成之后进行。例如性能测试，在单元测试或集成测试阶段也可以对被测单元或被测组件进行部分的性能测试。但这些都不能作为系统性能指标的依据，所以在系统测试阶段所做的性能测试是必不可少的。
- 系统测试是在模拟真实环境下的测试。相比之前的测试阶段系统测试所使用的系统配置、软硬件环境都必须按照产品需求规格说明书中的要求严格进行配置。因此，系统测试阶段出来的结果会更具有现实意义。
- 某些特殊的常量数据只能在系统测试阶段来验证，例如关于用户界面友好程度、UI 的美观程度、字体的大小以及字体的类型等。

面试试题2

系统测试是为了验证哪一个文档？

解答

在软件的开发过程中，开发人员通常需要将软件的市场需求说明转化为软件的产品需求规格说明书，然后再依次向下细化为软件详细设计规格说明书。在产品需求规格说明书中会指明该软件产品要实现哪些功能、不需要实现哪些功能、要处理哪些特殊的异常、产品的各项性能指标以及产品工作的软硬件环境等。系统测试就是对应于该文档，对软件进行全面的系统的测试。

面试试题3

什么时候开始做系统测试？

解答

系统测试活动应当在产品的所有组成模块都集成了之后进行。这就意味着该产品的所有模

Offer!

块都顺利地完成了集成测试，并通过了集成测试审查，符合集成测试的出口标准。

面试试题4

怎么做系统测试？

解答

产品需求规格说明书中通常包括对产品功能的需求描述，除此之外，还会有对产品性能、安全、健壮性及用户易用性等要求。因此，在系统测试阶段通常都包含有以下测试。

- 功能测试：测试产品的各项功能的正确性，以及验证产品的各项功能是否与产品需求规格说明书描述的一致。

功能测试应当首先保证的是功能的正确性。如果发现功能实现的与产品需求规格说明书描述的一致，但是并不正确，那么测试人员应当及时地与开发人员代表、项目管理人员和需求管理人员沟通以确定产品需求规格说明书中对被测功能的描述是否是一个缺陷。如果是，则应当将其记录为一个需求文档的缺陷，并予以跟踪。

- 性能测试：测试产品的各项性能是否达标。

例子

- 例如在手机测试中，测试手机在睡眠待机、非睡眠待机、屏幕亮起等情况下的电流安培数及手机待机时间等指标是否达到预定值。
- 又例如在网站测试中，系统能并发处理的用户请求的数量等是否达到预定值。

- 压力测试：测试以获得产品的某些关键性能指标的极限。

例子

- 例如在手机测试中，手机最长待机时间、手机连续播放视频的时间、手机打开一条彩信所花的最短时间、手机拍摄并存储一则大容量的视频所花的最短时间等诸如此类的性能指标的极限值。
- 又例如网站测试中，系统能并发处理的用户请求的数量的极限值。

这些极限值意义在于，系统设计人员可以通过这些数值预测系统在何时需要升级。

- 健壮性测试：指软件产品正常运行时在遇到某些突发异常事件的系统保持与恢复的能力。

例子

例如网站服务器在突然断电或电缆被切断的情况下备用服务器是否能正常启动并接受用户请求。

- 兼容性测试：指软件产品的新旧版本之间的兼容以及软件产品与软件和硬件环境间的兼容。

Offer

例子

- 例如很多大公司在升级某种广泛使用的办公软件时（如 Word），都会先做一下兼容性的测试以免产生数据的丢失。这其实就是一种新旧版本兼容性的测试。
- 又例如在测试网站的过程中，通常要测试网页是否能在其他操作系统上的其他浏览器中正确显示。常用的操作系统有 Windows、Linux、Mac OS 等，每种操作系统又有多个版本，而常用的浏览器有 IE，Firefox、Chrome、Netscape 等，而且同样有很多的版本。这样，要想在所有的操作系统/浏览器组合上测试网站的兼容性那几乎是不可能的，此时就必须有其他的策略，例如着重挑选网站客户经常使用的操作系统（常用版本）/常用浏览器（常用版本）的组合作为备选组合，然后再使用正交试验设计法产生测试用例。

- 用户易用性测试：指软件产品是否为用户提供了友好的界面及易操作的接口。

例子

- 例如在手机测试中，一些常用的重要的手机功能（如电话簿、短信等）是否能在很少的步骤（有的公司要求为 3 步）内完成一项操作（如存储一个通讯记录或删除一条短信等操作）。
- 又例如在网站测试中，网页的美观和一致性等。例如在微软公司的网站上，几乎所有英文字符的字体都默认统一被设为“Verdana”。

- 安装测试：软件产品安装是指测试软件在各种软硬件环境组合下是否能正常地进行安装、卸载。

测试安装时还包括取消安装时的回滚操作是否彻底、是否清除了文件系统和注册表中的内容以及在安装过程中出现异常（断电、系统崩溃等），待到系统重启之后软件产品是否可以重新安装等。

警示

软件的安装测试是非常重要的测试。历史上不乏因安装测试做得不够彻底而产生大量顾客退货的情况。

面试试题 5

系统测试的入口条件和出口条件是什么？

解答

入口条件（Entry Criteria）。

在做系统测试之前都需要做一些准备工作，其包括。

- 软件产品的集成测试测试结果已通过并被检视。
- 系统测试计划已被检视并通过。

Offer!

- 系统测试策略已被检视并通过。
- 系统测试测试用例已开发完成并被检视通过。
- 所需的工具代码已完成并被检视通过。

出口条件（Exit Criteria）。

系统测试完成后如果完成了以下工作就可以进行下一阶段的测试。

- 所有的系统测试用例都被执行。
- 系统测试测试覆盖率达到了预定的要求。
- 软件产品的缺陷率低于预定的上限。

3.8 用户验收测试

什么是用户验收测试?

当软件产品顺利通过了系统测试之后就开始了用户验收测试阶段。用户验收测试，顾名思义就是交付给用户，让用户执行产品验收所进行的测试，因此也被称为交付测试。用户验收测试是产品上市之前的最后一个正式的测试阶段。

下面列举几道典型面试题来帮助大家理解。

面试试题1

为什么要对软件进行用户验收测试?

解答

每一个产品都是为最终的用户设计和开发的，因此让用户参与测试是获取用户产品反馈的重要途径。

通过用户验收测试可以实现在用户环境中对产品规格说明书进行验证。

通过用户验收测试可以实现在用户环境中对产品的用户文档进行验证。

通过用户验收测试可以降低产品中的缺陷数量。

通过用户验收测试可以获取用户关于产品功能、性能的反馈，有利于产品团队在产品上市之前对产品的用户印象产生一定认识。

面试试题2

如何开展用户验收测试?

解答

虽然用户验收测试是让用户执行产品验收所执行的测试，但是由于各个项目的开发目的和

Offer!

用户群体的千差万别，其在执行上也会有很大的差别。

对于一些用户群体是企业内部人员的软件项目，规划和执行用户验收测试是比较容易做到的。通常测试组织者会在企业内部挑选参与测试的人员、搭建测试环境、分配测试用例，然后对测试执行情况进行跟踪。

对于另外一些用户群体是其他企业或团体的软件项目也可以按照上述的方法进行计划和执行。另外，还必须安排软件生产团队的内部测试人员对用户体验测试的参与者给予一定的帮助，并对用户提出的反馈和报告的软件缺陷进行整理。

对于提供大众型服务的产品，例如手机、免费邮箱等，其用户体验测试就扩展成为 Alpha 测试和 Beta 测试。

说明

Alpha 测试：是指产品上市之前在公司内部进行的用户体验测试。在这类测试计划下，测试任务可以分配给有经验的测试人员，产品的设计、开发部门，市场部门，以及售后支持部门、培训部门等。对参与测试的专业测试人员可以指定专门的测试用例，以方便了解在用户环境下这些测试用例的执行情况。其他参与测试的人员可以在使用过程中对其进行随机测试（Adhoc Test）。在很多大型的面向大众消费者的企业 Alpha 测试是其大众型服务的产品项目完整测试计划的一部分。

Beta 测试：是指产品上市之前让用户或企业的合作伙伴进行用户体验测试的行为。在这类测试计划下，测试的组织人员主要工作难点为对参与测试的用户进行挑选，搭建测试环境和建立有效的用户反馈收集渠道。而参与测试的人员大多是在产品使用过程中对其进行随机测试（Adhoc Test）或提供用户角度的意见。现在很多公司的免费产品都对用户宣称是 Beta 版本，而不是正式版本，有的甚至提出永远是 Beta 版。

具体细节可参看本书第 3.4.7 小节的相关内容。

用户验收测试所要做的通常是在用户环境中对产品规格说明书进行测试和验证。除此之外，测试人员还需要对用户文档进行测试，包括帮助文档、用户手册、安装配置指南、多媒体演示资料、授权使用提示、软件包装及广告等。对文档的测试主要针对文档的准确性、完整性和通俗易懂性。

面试试题 3

用户验收测试验证的是哪个文档？

解答

正如前面所说的，用户验收测试所要做的通常是在用户环境中对产品规格说明书进行验证。测试人员所要做的是逐字逐句按照产品规格说明书对产品进行测试，这就需要测试用例设计人员要把握产品规格说明书的细节来设计用户验收测试的测试用例，并且确保使用的产品规格说

Offer!

明书的版本是最新的版本。

面试试题4

开展用户体验测试收测试？



答

1. 明确用户体验测试的目的

- 通过用户体验测试使产品团队在产品上市之前可以了解用户的反馈，从而针对有价值的反馈对产品进行必要的更新。
- 通过用户体验测试得到来自于用户角度的缺陷报告，这样可以降低由于生产者和消费者各自观点和角度不同而产生的产品缺陷数量。

2. 用户体验测试的准备工作

- 挑选参与用户体验测试的用户。首先，在这个过程中挑选用户应当针对产品的目标群体进行挑选，应尽量覆盖目标群体中的各个年龄、收入、教育水平等层次。其次，在挑选过程中还应当注意被挑选的用户不能具有该产品研发背景的专业人士。例如，如果被测产品是手机，那么参与用户体验测试的用户就不应该有研发手机的背景。
- 搭建测试环境。用户体验测试开始之前产品团队先要决定使用哪一个版本的产品。如果是网络服务性质的产品则必须搭建好参与用户体验测试的用户可访问的测试环境。如果是单机版软件，则必须提供下载环境或可安装光盘。如果是手机，则必须确定参与测试的软硬件版本，并将软件导入到测试手机中，同时还要对手机做一些有效性的验证，以证明其能正常使用。
- 分发被测产品。产品团队应负责将被测产品分发给参与用户体验测试的用户，例如分发网站账户、提供产品下载账户、提供产品安装光盘、提供手机终端等。
- 建立用户与产品团队的沟通渠道。进行用户体验测试的目的是为了获取用户的使用反馈意见，因此便利有效的沟通渠道是用户体验测试成败与否的关键因素之一。通常产品团队都会建立一个方便参与用户体验测试的用户反映产品问题和提出意见的网站。用户在使用产品的过程中如果发现了问题或有好的意见，可以通过用户体验专用网站把信息传递给产品团队。

3. 用户反馈的过滤

前面我们提到在用户体验测试的准备工作中必须建立用户与产品团队的沟通渠道，而这个渠道通常是用户体验专用网站。用户在使用产品的过程中如果发现了问题或有好的意见可以通过用户体验专用网站把信息传递给产品团队。这些信息一般被总结为来自于用户角度的缺陷报告（Customer Defect）和改进意见（Customer Enhancement）。然而这些用户信息有时会出现描述不够完全、表述不够清楚或信息重复等情况，因此所有从用户体验专用网站收集的信息都要经用户体验缺陷管理人员的过滤。之后再形成有效的产品缺陷或产品改进，并将其录入到产品

Offer!

缺陷管理跟踪系统中，这样才能减少由于用户体验缺陷报告或改进意见不够专业而带来的产品开发人员的额外工作量，如图 3-14 所示。

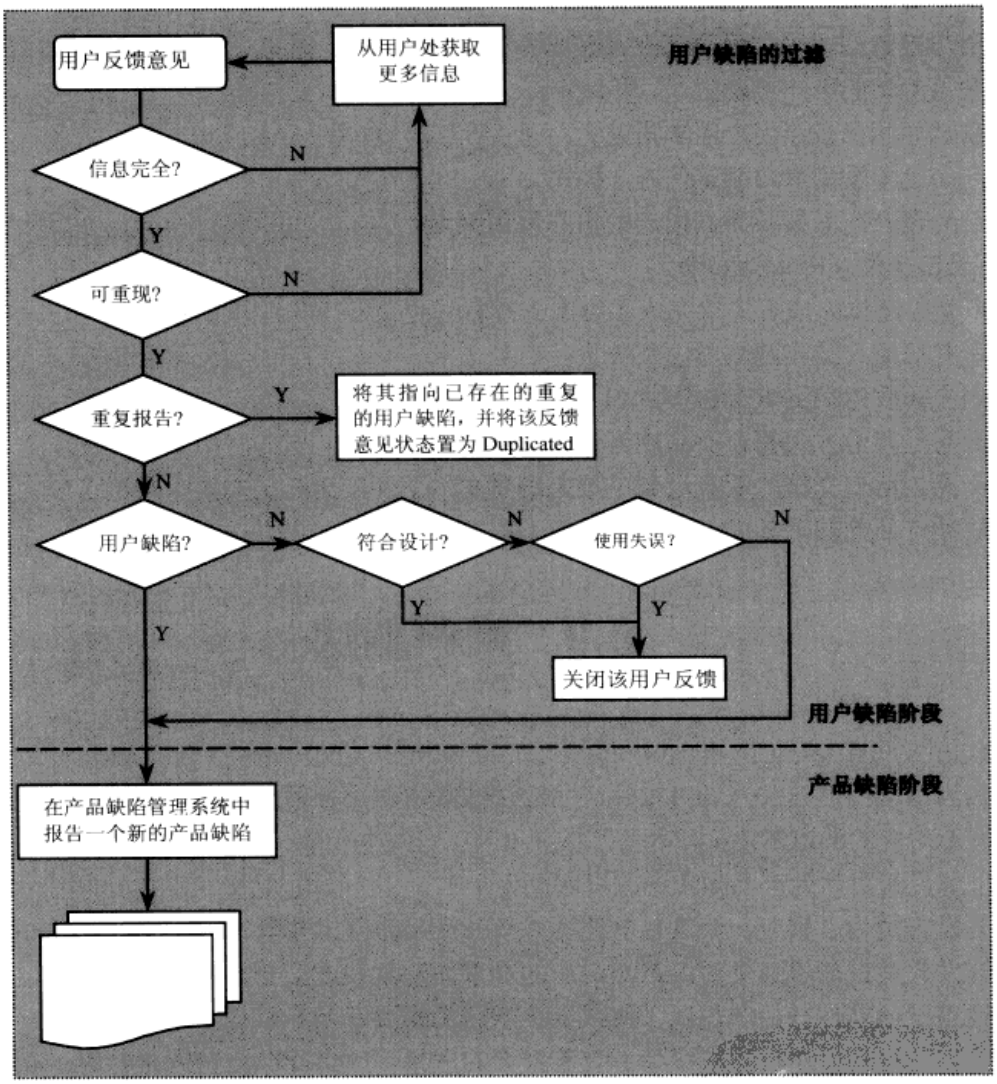


图 3-14 用户缺陷的过滤

面试试题 5

用户验收测试的出口条件和入口条件是什么？

解答

1. 入口条件（Entry Criteria）

在做用户验收测试之前都需要做一些准备工作，包括。

- 软件产品的系统测试测试结果已通过并被检视。
- 用户验收测试计划已被检视并通过。
- 用户验收测试策略已被检视并通过。
- 用户检验测试测试用例已开发完成并被检视通过。
- 参与测试的用户已确定。
- 测试环境已按照用户需求搭建。
- 产品已分发到参与测试的用户手中。
- 已经建立产品参与测试用户的信息反馈渠道。

2. 出口条件（Exit Criteria）

用户验收测试完成后如果完成了以下工作就可以进行下一阶段的测试。

- 所有的用户验收测试用例都被执行。
- 用户验收测试测试覆盖率达到了预定的要求。
- 产品的行为方式符合产品规格说明书的要求。
- 产品的用户文档的质量达到了预定的要求。
- 软件产品的缺陷率低于上市限制的上限。

3.9 回归测试

什么是回归测试？

回归测试就是当开发人员对软件产品的基线版本做出任何改变时，测试人员针对这些改变进行的有针对性的测试活动。

以上所说的对软件产品做出的改变包括：

- 开发人员为了修复一个软件缺陷而对软件产品进行的修改。
- 开发人员在需求变更时，为满足新的需求而对软件产品做出的修改。
- 在稳定的软件产品中加入（集成）一个新的模块。

回归测试与软件产品的基线版本有着很大的联系，因为回归测试就是针对于软件产品的基线版本的改变所做的测试。

面试试题1

什么是软件的基线版本（baseline）？

解答

软件产品的基线版本就是软件在开发的过程中某一时刻的快照（snapshot）。在这一时刻的软件产品的版本一定是一个稳定的版本，一定可以作为继续在其上做开发的依据。



此后开发人员要继续开发就必须在基线版本中拉出一条分支（branch）。待新的功能开发完成并经过测试稳定之后，再经过项目管理人员、开发团队负责人、测试团队负责人、软件配置管理人员（SCM）检视通过之后形成新的基线版本。需要注意的是，新的基线版本同样必须是稳定的。

如果新的基线版本在后续的开发及测试中发现有不稳定的现象甚至是影响软件功能或项目进度的严重的设计缺陷或软件缺陷，则此时新的基线版本可能就会被抛弃。而产品将会会滚（rollback）到之前的那个稳定的基线版本。当然，这种现象不是经常发生的，也是项目相关人员最不愿意看到的。

基线版本的来源：有些软件项目是从零开始做起的，其软件产品通过逐步的集成会形成一个包含主要模块并且逐步趋于稳定的版本。当其版本足够稳定时，软件配置管理人员会将其编译为基线版本，并通知项目中的其他角色项目的基线版本已经产生。此后的开发都基于此版本。

有些软件项目是基于以前较成功的产品进行开发的，在这种情况下可以挑选以前产品中较稳定并兼具扩展性的版本作为新产品的基线版本。此后的开发都基于此版本。

面试试题 2

为什么要做回归测试？

解答

因为回归测试就是针对于软件产品的基线版本的改变所做的测试，因此通过对产品进行回归测试可以。

- 验证开发人员所承诺修复的软件缺陷是否已被正确修复。
- 验证新的软件修改是否影响了原有的稳定模块的正确性和稳定性。
- 验证新的软件修改是否引入了新的缺陷。
- 验证新的软件版本是否稳定，从而成为新的基线版本，以备后续开发使用。

面试试题 3

什么时候开始做回归测试？

解答

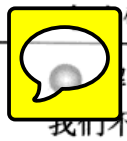
从回归测试的定义（回归测试就是当开发人员对软件产品的基线版本做出任何改变时，测试人员针对这些改变进行的有针对性的测试活动）可以看出，首先正在开发的软件产品必须形成了基线版本。当然，这个基线版本可以是某一个模块的基线版本（即产品的某一个模块的版本趋于稳定，此后在这个模块上的开发都基于该版本），也可以是整个项目的基线版本。

另外，还必须要有对基线版本的修改。此时软件配置管理人员（SCM）会将修改后的代码编译成新的版本，然后分发给测试部门进行回归测试。

Offer!

面试试题4

做回归测试？



答

我们不难看出回归测试是贯穿于集成测试、系统测试和用户体验测试等测试阶段的一种测试行为。它的目的是为了评估在以上阶段中修复软件缺陷、引入新的模块等行为对整个基线版本的质量产生的影响以及新的修改自身的质量。

下面以系统测试阶段的回归测试活动为例来讲述怎么做回归测试。

正在开发的软件产品进入系统测试阶段之后，软件配置管理人员（SCM）就会为这个可以进行系统测试的版本打上系统测试阶段的基线版本标签（Label），而系统测试部门就可以开展系统测试了。

此时测试人员按照事先开发完成的测试用例及测试脚本对此基线版本进行测试。测试人员发现疑似软件缺陷后对其进行复查并确认，如果是软件缺陷，就将其记录在缺陷跟踪管理系统中以备跟踪。这时测试人员、开发人员、项目管理人员和软件质量管理人员就会对缺陷的严重性和修复的优先级进行定级。在这一轮（cycle）的系统测试结束之后，以上人员将对在缺陷跟踪管理系统中的所有在这个基线版本上发现的缺陷进行讨论，以确定所有缺陷的修改进入到软件新版本的日程。

同时，需求人员、测试人员和开发人员等还有可能提出新的软件功能的需求。这些新的功能扩展的描述也将被记录在一个跟踪系统中（有时和缺陷跟踪管理系统是同一系统）。这时该需求将被需求人员、测试人员、开发人员、项目管理人员和软件质量管理人员进行讨论并评估其风险（该风险包括计划进度的风险及引入新功能对原有功能模块的影响）。如果讨论和评估通过，则开发人员就可以开始对新的功能进行开发。待新的功能模块经过测试并且稳定之后就可以在某个时刻集成入原有的软件系统中。

当以上两者情况的修改都确定要进入新的软件版本时，测试人员就需要对这些修改设计新的回归测试计划。

设计回归测试计划的策略。

- 在设计新的回归测试计划时，可以将用于覆盖上一基线版本的所有测试用例加入到新的回归测试计划中。之后，还必须加入针对新功能模块开发的新测试用例。这种策略的优点是测试用例覆盖率最高，缺点是执行回归测试计划人员的工作量巨大。
- 另一种策略是在新的回归测试计划中只包括对所有软件缺陷修复的验证及对新加入功能的系统级的验证。这种策略的优点是执行回归测试计划人员的工作量最小，缺点是测试用例覆盖率最低。
- 最后一种策略是在新的回归测试计划中包括对所有因软件缺陷修复而产生的修改的覆盖。这就意味着除了验证软件缺陷的修复还必须评估这些修复对原有的稳定的模块的影

Offer

响。因此，还必须加入一些用于验证与被修复的缺陷有密切关系的模块稳定性的一些测试用例。这些测试用例可以是原有的测试用例，也可以是测试人员新开发的用例。同理，对于新加入的功能，除了验证其本身还要对可能被其影响到的模块进行再一次的验证。这种策略的优点是执行回归测试计划人员的工作量较小，测试用例覆盖率较高。缺点是新的回归测试计划的质量和测试覆盖率受开发该计划的人员的经验、能力的影响较大，此外开发新的回归测试计划所需的工作量也较大。

此后测试人员又回到了测试的执行阶段。待测试完成之后，测试负责人给出测试报告以反映软件的质量。当软件的质量还不够稳定时，就将进入下一轮的缺陷修复与回归测试。如果软件的质量趋于稳定时就可对其进行检视以确定软件是否符合系统测试的出口要求以进入软件测试的下一阶段。

我们不难看出对软件进行持续的回归测试是使软件质量趋于稳定的一个重要手段。在一轮又一轮的回归测试中软件的缺陷数量应该是一个逐渐收敛的趋势（在不加入新的功能并不引入大量新缺陷的情况下）。当缺陷数量或比例到达某一个预先设定的值时，我们就认为该软件的质量已经满足这一测试阶段的出口要求了。此时，有的公司还会对产品软件产品进行一次全回归测试（Full Regression Test）。只有当全回归测试计划中包括能够覆盖整个软件产品的所有测试用例时，这个全回归测试的测试报告才能被用来作为认定该软件的质量是否已经满足这一测试阶段的出口要求的依据。

3.10 冒烟测试

冒烟测试（Smoke Test），这一术语源自硬件行业，是指对一个硬件或硬件组件进行更改或修复后直接给设备加电，如果没有冒烟，则表示该组件就通过了测试。

软件测试中的冒烟测试，就是指在每个开发部门开发出的版本建立后，由测试部门或集成部门对系统的主要的基本功能进行简单的测试。这种测试主要是强调功能的覆盖率，而不侧重验证功能的正确性（不侧重但不意味着不在意）。如果这些测试用例都能通过，则正式进入下一阶段的测试。

例子

当时摩托罗拉公司的 Linux-Java 手机研发系统拥有多个不同的测试部门，笔者当年是在平台测试（Platform Test）部门。产品进入平台测试之前，会由产品发布测试（Product Release Test）部门进行冒烟测试，再交付给平台测试部门。这种冒烟测试在摩托罗拉公司称为 Sanity Test。

在冒烟测试中应注意。

（1）只覆盖主要功能。

冒烟测试不需要面面俱到，只需将重点放在保证主要功能或主要业务逻辑路径执行正常即可。

Offer!

（2）易用性。

自动化测试脚本运行方式简单，应避免再让执行人员做各种配置。

（3）配置管理。

脚本程序、脚本设计、脚本需求、脚本更新日志及测试用例等文档要进行配置管理。

（4）测试脚本要独立。

也就是经常说到的“高内聚，低耦合”的思想，每个测试脚本要尽可能的独立，执行过程不需要依赖其他测试脚本。

（5）测试脚本要简单。

测试脚本编码要尽可能的简单，节省人工同时避免脚本出现 Bug。

（6）测试结果比对。

测试结果要留存。对比一段时间内的测试结果，用于判断软件哪里存在问题。

通常面试题范围都集中在概念和项目经验两方面。

下面列出 5 个常见面试笔试题，以资读者参考。

面试试题 1

冒烟测试用手工执行好，还是用自动化脚本好？

解答

各有不足。所有的异常情况很难在自动脚本中指出，因此自动化有可能会漏掉 Bug。但是有的功能、性能无法用手工方法测，只能使用自动化脚本。自动化测试速度通常快于手工测试。在实际应用中，一些简单功能可以使用自动化脚本完成，而对于较复杂的功能最好使用手工测试。

例子

在摩托罗拉公司的 Linux-Java 手机研发系统，在产品发布测试部门测试通过软件版本之后，平台测试部门仍要内部进行冒烟测试，来决定是否正式接受这个软件版本。测试方法是采取手工结合自动化脚本。比如一些复杂通话功能的测试是用自动化脚本完成。这种测试基本不可能用人工方法实现。

面试试题 2

冒烟测试是“接受/验收测试（Acceptance Test）”吗？

解答

我们可以认为冒烟测试是一种特殊的接受/验收测试。它的独特之处在于测试用例侧重功能的覆盖率，而不侧重功能的正确性，而一般的接受/验收测试的测试用例同时也要求功能的高正确率。

Offer

面试试题3

如果冒烟测试没有通过，这个软件版本会被如何处理？

解答

会和测试报告一起被退回开发部门。这意味着，冒烟测试中有一定量测试用例没有通过，低于被测试部门正式接受的标准。开发部门在解决问题后，自测中通过了这些上次未通过的用例之后，会再次交付软件要求冒烟测试。

面试试题4

如果冒烟测试中只有1条测试用例没有通过，也会被退回开发部门吗？

解答

不一定，但很可能会被退回。通常会有通过率标准，尽管一般是要求100%通过。毕竟这些测试用例都覆盖的是非常基础的功能。若连这些测试用例都无法通过，则后面对细节的测试中会产生大量的Bug。

面试试题5

为什么要进行冒烟测试？

解答

主要是为了节省测试部门的人力。开发部门被要求负起责任，尽量不要产生一些严重的问题。例如某个需求中的功能没有覆盖到，当这些严重问题在正式测试中才被发现，就会造成更大的人力物力开销。因此，冒烟测试就是针对这种情况被制订出来的，保证交付给测试部门的软件已有了一定的质量。实际上，开发部门在开发时就参照了冒烟测试的测试用例。

3.11 α、β、γ测试

α、β、λ测试是3个不同的测试阶段。

α和β测试属于产品验收测试范畴，是在测试部门结束全部测试之后进行的。

α测试是第一阶段，一般只供内部测试使用。

β测试是第二阶段，软件中大部分的不完善之处已被消除，但仍有可能还存在缺陷和漏洞，一般只提供给特定的用户群来测试使用。

λ测试实际上不完全是软件测试了，而主要是市场测试，是第三阶段，此时产品已经相当



成熟，投放于小块区域的市场。只需在个别地方再做进一步的优化处理就可以全面上市。

在α、β、λ测试中都有可能发现软件缺陷。

下面通过几道典型例题来考查一些基本概念，这些概念在面试中常常碰到。

面试试题1

什么是α测试？

解答

α（Alpha）测试是由用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。α 测试的目的是评价软件产品的功能、可使用性、可靠性、性能和产品支持能力，尤其注重产品的界面和特色。α 测试可以从软件产品编码结束之时开始，或在模块（子系统）测试完成之后开始，也可以在确认测试过程中产品达到一定的稳定和可靠程度之后再开始。需要注意的是，α 测试不能由程序员或测试员完成，而在 α 测试中发现的错误，可以在测试现场立刻反馈给开发人员，由开发人员及时分析和处理。

α 测试的关键在于尽可能逼真地模拟实际运行环境和用户对软件产品的操作并尽最大努力涵盖所有可能的用户操作方式。软件产品经修改通过α测试后称为β版本。

面试试题2

什么是β测试？

解答

只有当α测试达到一定的可靠程度时，才能开始β（Beta）测试。同时，产品的所有手册文本也应该在此阶段完全定稿。与α测试相同，β测试也不能由程序员或测试员完成。

一般而言，β测试就是由一部分受控制的客户进行的黑盒测试。

β测试是由软件的多个用户在实际使用环境下进行的测试，这些用户返回有关错误信息给开发者。在β阶段，用于β测试的产品被发布到一部分受控制的公司外部人员手中，通过这部分受控制的外部人员的测试和反馈，β阶段可以尽量发现产品中存在的缺陷和错误。在某些情况下，β版本可能被发放到范围更广的外部人员手中（例如，通过网站下载或是其他方式面向公众发放）。测试时，开发者通常不在测试现场。因而，β测试是在开发者无法控制的环境下进行的软件现场应用。在β测试中，由用户记下遇到的所有问题，包括真实的以及主观认定的，然后向开发者报告。β测试主要衡量产品的特性，着重于产品支持能力，包括文档、客户培训和支持产品生产能力等。

由于β测试的组织难度大、测试费用高、测试随机性强、测试周期跨度较长以及测试质量和测试效率难于保证，使得很多软件可能不进行β测试。随着测试技术的提高和专业测试服务机构的大量涌现，很多软件的β测试被外包给这些专业测试机构进行测试。

Offer

面试试题 3

什么是 γ 测试？

解答

γ（Gamma）测试是一个较少被提及的测试阶段，也称为实际市场测试（In-Market Testing）。

⚠ 警示

不要与“测试市场（Test Marketing）”相混淆，后者是指考察产品的可销售性和在市场上的生存能力。

γ 测试在 β 测试通过后进行，共目的是通过投放产品在小块区域来评估产品本身和其市场计划。

通过真正的销售渠道，产品在一段时间内被销售在 1 个或两个小地理区域，同时配有一定的销售策略，如投放广告、给予特价等。

这种市场测试将能够更准确地预测出单位销售量，并找出全面投放市场计划的不足，从而对其进行调整。

面试试题 4

β 测试等价于黑盒测试，这样说对吗？

解答

对 β 测试常见的一个认识误区就是“β 测试等价于黑盒测试”。实际上，β 测试对应在软件产品发布之前的 β 阶段，而白盒、黑盒和灰盒测试技术是从技术和方法层面对测试的描述，不应该将这两部分概念混淆。

面试试题 5

α、β、λ 测试是由软件测试部门来做吗？

解答

不是。α 测试通常由公司内部的非测试人员来进行，β 测试通常由公司外部人员来进行，λ 测试主要是小区域的市场测试。

3.12 性能测试 1——性能指标

要进行性能测试，就必须知道有哪些性能指标需要做测试。这些性能指标从其本质来说都体现在对时间的利用和对资源的占用上。

Offer

我们都希望做出来的软件能够满足客户最高的时间效率和最低的资源要求。

本节我们将列举部分性能测试的指标。

面试试题1

常用的性能测试指标有哪些？

解答

常用的性能指标通常有软件的事务处理效率、IO 性能、数据库性能、内存利用率和 CPU 利用率等时间与资源的使用效率。

面试试题2

什么是软件的事务处理时间？

解答

软件的事务处理时间是指软件在处某一项事务时，从开始处理到完成所花费的所有时间。该时间值不但包括软件产品本身处理该事务所花费的时间，而且还包括使用者与软件产品的人机交互所花费的时间。

测试软件产品本身处理该事务所花费的时间是指在某一固定配置下，测试软件产品从开始处理事务到处理完成所花的时间。

分阶段测试软件产品对某一事务的处理时间，找出其中的瓶颈路径。例如在数据转换和传输过程中，某软件产品采用了多线程的数据转换处理方式，但在传输时由于架构的原因采用的是单线程的发送方式。这样该软件产品的数据传输发送环节就成为了其瓶颈。

找出软件的最大事务处理时间。

使用者与软件产品的人机交互所花费的时间属于软件易用性测试的范畴，但实际上它在软件的事务处理测试中是非常关键的一环。因此我们需要测试用户在与软件产品一起完成一项事务时所花费的平均时间长度。

例子

其实很多图形化应用软件提供商都对软件的易用性提出了一些硬性的指标。例如，在完成一项功能时，用户的控制性操作步骤不得超过 5 步；又如手机中一个应用程序的弹出窗口深度不得超过 3 个等。这样的目的都在于提高人机交互的效率，赢得市场。

面试试题3

什么是软件产品的 IO 性能？



解答

IO 性能通常包括物理硬盘的 IO 性能、网络的 IO 性能和其他硬件的 IO 性能。

以物理硬盘的 IO 性能为例。在需要大量进行读写硬盘操作的软件产品中物理硬盘的 IO 性能很容易成为其瓶颈，例如多任务多线程的下载大量文件等。

衡量 IO 性能的指标是单位时间内读写数据的数量，例如物理硬盘每秒钟读取或写入数据的字节数。但硬盘设备多年来的发展都集中在其容量上，其数据读写还都是机械式的，因此很容易成为系统中的瓶颈。

作为软件产品本身可以通过降低读写硬盘的次数，提高每次读写数据量大小来降低硬盘 IO 性能对软件性能的影响。例如，可以在批量处理数据时，等结果数据累计到一定数量时再将其一次性写入硬盘中。

通常在白盒测试中比较容易实现对 IO 性能的测试，而在黑盒测试中也可以进行 IO 性能测试。单通常数据的误差较大，其好处就是可能发现仅依靠白盒测试无法发现的缺陷。例如，在白盒测试时没有出现对某一文件的读写权限的依赖，而在黑盒测试中出现了。

面试试题 4

什么是数据库性能？

解答

数据库性能是指软件在进行数据库的查找、添加、删除、更新数据记录时所花费的时间和资源，例如网站测试中添加一条用户记录所花的平均时间等。

提高数据库访问性能的方式通常有。

- 减少对数据库操作的次数，把一些常用数据长期储存在内存中。
- 减少表与表之间的依赖性，避免在查找、添加、删除、更新数据记录时要同时对多张表进行操作。

同样在白盒测试中比较容易实现对数据库性能的测试，而在黑盒测试中也可以进行数据库性能测试。黑盒测试中可以模拟基于用户操作的大量对数据库操作的请求以测试软件本身和数据库的应对能力。

面试试题 5

什么是资源利用率？

解答

资源利用率就是软件产品实际使用资源数量与所申请的资源总量之间的比例。

内存空间利用率：即实际使用的内存空间和软件所申请的内存空间的比例。例如，用数组存储字符时，申请的字符数组长度为 1000，而实际使用的为 30。

Offer

CPU 的利用率：即应用程序使用 CPU 时间与应用程序总的运行时间的比例。

通常在分时系统中，客户端的软件 CPU 的一般不能太高而影响其他软件的运行，而在服务软件中，一般要求软件的 CPU 利用率较高以提高该软件的资源利用率。

3.13 性能测试 2——负载测试

负载测试是通过对被测软件产品进行各种方式的加载负荷以获得被测产品在各种加载情况下的性能参数等信息。

本节将讲述负载测试的目的、开始的时间、配置和加载策略等知识。

面试试题 1

负载测试的目的是什么？

解答

（1）对被测产品各性能参数进行不同频率、不同时间长度及不同负载量的加压测试。以便于项目的开发、测试人员了解产品的性能瓶颈。

例子

在对被测产品进行整体性加压时，我们可以得到一组产品处理能力的上限指标值。之后可以分别对产品的各项指标进行针对性的加压以得到各项指标的实际上限。然后将单项的性能指标上限与整体性能测试时的对应性能指标进行对比，我们就可以得到被测产品的性能瓶颈。之后项目的开发人员就可以根据负载测试中所得到的性能瓶颈对被测试产品进行改进。

（2）获得产品的某些关键性能指标的极限。以便于产品设计人员对产品的升级周期进行预测。

例子

新开发的软件产品升级版本通过负载测试验证其处理能力为每秒钟可以处理 10 万条数据。而对于现有的运营系统，平均每秒需要处理的数据量为 2 万条。峰值速率为 5 万条数据每秒。这样我们就可以以需要处理的数据量的历史增长记录为依据对何时需要进行下一次软件的升级做出预测。

（3）通过压力测试的结果获得产品的极限值，再以这些极限值作为参考得到产品的期望工作条件值。此后测试人员就可以使用这些期望工作条件值对产品进行性能测试和浸透测试等。

例子

新开发的软件产品升级版本通过负载测试验证其处理能力为每秒钟可以处理 10 万条数据。在获得极限值后我们可以根据以往的统计比例值得到一个期望稳定运行值（一般为极限

Offer!

值的 2/3）。再将此值作为性能测试、浸透测试等测试的设定值。

面试试题 2

何时进行负载测试？

解答

非功能性测试需要在产品的基本功能稳定之后才能开展，因此该测试一般都被安排在产品的功能测试进行了一段时间之后以及产品的各项基本功能稳定之时。此时非功能性测试的运行不会受到功能不健全或功能性错误的影响。

在非功能性测试开始之时，就应该立即开始对产品进行负载测试，因为其他性能测试大都需要通过负载测试来产生其参考值。

面试试题 3

负载测试需要做哪些环境配置？

解答

系统测试对环境要求很高。各项软硬件、网络等的配置都要接近甚至达到产品真实运行环境的水平。负载测试作为系统测试中非功能测试的一种，其对产品的环境配置也会有很高的要求。

其硬件配置包括主板型号、CPU 型号、内存大小及型号、硬盘大小及接口类型、网卡的接口类型及支持的带宽。此外，还需要考虑一些其他专用设备的配置，例如负载均衡器、交换机、代理服务器等。

软件的配置包括服务器的操作系统、杀毒软件、数据库软件、Web 服务环境及业务逻辑服务环境等。

除软硬件配置外，在负载测试中还必须用到一些自动化的测试工具来模拟用户的请求和网络的状态等参数。例如，可使用 TCPReplay 来播放一些预先抓取的数据包；用 WireShark、TCPDump 等工具在网络上抓取用来回放的数据；或用来抓取进出被测产品的数据进行对比以测试产品对数据处理的正确性及有无丢包。另外还有一些网络状态的模拟工具可以模拟网络的延时、误码、各种带宽、数据包掉包、出现重复数据包及数据包顺序混乱等情况或现象。这些自动化工具大多都是手工测试难以模拟的，因此在负载测试环境搭建时，必须将这些测试工具安装在适当的位置，并将其调试成功。

搭建并调通负载测试的测试环境通常是一件非常辛苦且繁杂的工作，因此测试计划人员一定要事先做好计划、预留出足够的时间来完成这项工作。

面试试题 4

负载测试的加载策略有哪些？

Offer!

解答

- 一次性加载：即一次性为被测产品加载定量的负载，并且维持一个较长的时间段。该加载策略通常用于验证被测产品在某一固定负载下是否能长时间的正常工作。
- 递增加载：即逐步为被测产品增加负载，直到被测产品不能正常工作为止。该加载策略通常用于获取被测产品正常工作的负载极限。
- 峰谷试加载：即为被测产品加载一个较高的负载，并且维持一段时间，然后再为被测产品加载一个较低的负载（或把负载降为零），并且维持一段时间，如此不断重复。该加载策略通常用于发现被测产品在不断的申请资源、释放资源的轮回中是否会产生资源或内存的泄露。
- 随机加载：即随机为被测产品加载定量的负载，并且维持一个较长的时间段。该加载策略通常用于模拟真实情况下被测产品的运行状况，并可能发现一些不在测试用例设计范围内的产品缺陷或性能瓶颈。

3.14 安全性测试

安全性测试（Security Test）主要是指有关验证应用程序的安全等级和识别潜在安全性缺陷的过程。应用程序级安全测试的主要目的是查找软件自身程序设计中存在的安全隐患，并检查应用程序对非法侵入的防范能力此时安全指标不同，测试策略也不同。

优秀的计算机基础和动手实践能力是高级安全测试工程师的必备能力。有位专家说：不懂二进制数据就无从谈起安全技术。

专家中最精英的人士往往精通黑客技术，例如精通汇编语言和反汇编技术、精通二进制机器代码、精通密码学知识、了解 PKI 体系架构、擅于分析病毒样本。他们有能力拿掉 PE 上复杂的保护壳，并在二进制文件中定位漏洞。

事实上，若想彻底清除软件中的所有安全隐患基本上是不可能的。在工作和面试中，我们通常只需要注重一些主要的、常见的问题即可。

例子

常见的情况是，对于一个公司内部使用（只有通过内网才能够访问，在互联网上无法访问）的 B-S 结构软件，对这样的软件，在登录口令方面就不用过于注重安全性问题。例如密码可以是弱口令。如果这时你提交一些安全性软件缺陷，通常会被领导认为没有意义。

一些常见的软件安全问题有。

- 缓冲区溢出。
- 格式化字符串攻击。
- 整数溢出。

Offer

- SQL 注入。
- 命令注入。
- 跨站脚本攻击。
- 函数错误返回值、异常未被处理。
- 网络数据截获。
- Magic URL、隐藏的表单字段攻击。
- SSL、TLS 协议攻击。
- 脆弱的登录口令。
- 存储中的访问控制攻击。
- 系统和应用程序的信息泄露。
- 非法文件访问。
- 域名服务解析攻击。
- 竞争条件攻击。
- 未认证的密钥交换。
- 密码学随机数破解。

一些安全测试常用的静态检测工具有。

- Fortify。
- Rough Auditing Tool for Security。
- BEAM。
- SLAM。
- Flaw Finder。
- Prexis。
- Coverity。

安全性测试的主要方法。

- 静态的代码安全测试：主要通过对源代码进行安全扫描。
- 动态的渗透测试：模拟黑客的输入，对应用系统进行攻击性测试。
- 程序数据扫描：通常是进行内存测试。

正向测试过程是以测试空间为依据寻找缺陷和漏洞，而反向测试过程则是以已知的缺陷空间为依据去寻找软件中是否会发生同样的缺陷和漏洞。对于安全性要求较低的软件，一般按反向测试过程来测试即可，而对于安全性要求较高的软件，则应以正向测试过程为主，反向测试过程为辅。

- 反向测试过程的一个主要优点是成本较低，只要验证已知的可能发生的缺陷即可，但缺点是测试不完善，无法将测试空间覆盖完整，同时也无法发现未知的攻击手段。
- 正向测试过程的优点是测试比较充分，但工作量相对较大。

安全测试是很深奥的领域，想做得好，需要深厚的计算机知识。即使对于专业人士，也很



难认识到所有的安全漏洞。通常而言，对于普通的软件测试工程师，在这方面的面试题不会很深奥，因为工作中这方面的要求不会很高，但是大家仍要对其有一些基本的了解，并能够掌握一些主要的安全性缺陷的测试方法。

笔者限于水平，对安全测试领域认识有限，只有能力呈现一些基础的内容给大家，希望大家能继续钻研。

下面列出5个常见面试笔试题，以资读者参考。

面试试题1

什么是跨站脚本攻击？

解答

跨站脚本（XSS）漏洞，是指Web应用程序从恶意用户处得到输入，不用进行验证就直接回显（回显不必可见）。如果这种回显由恶意的可被浏览器解析的脚本代码生成，一旦受害浏览器单击了包含回显的链接，则受害浏览器将会执行这些恶意代码，从而受到攻击。

例子

在网页上，在任一可输入文本域输入：

```
<script>alert("siqi");</script>
```

单击“提交”按钮时，若弹出如图3-15所示的对话框，则说明存在XSS缺陷。

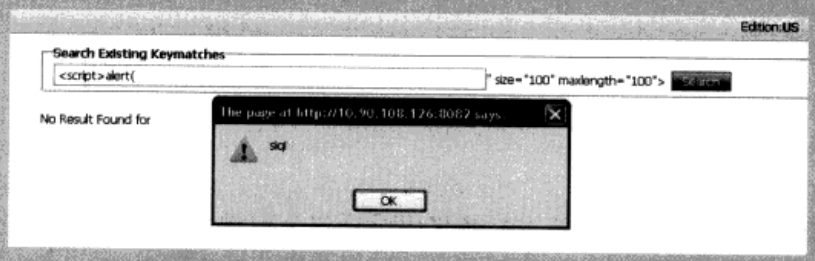


图3-15 XSS缺陷例1

如果另一个用户从另一台浏览器访问此网页，在没有输入恶意代码的情况下，当操作这个文本框时，若发现也弹出这个对话框，则说明恶意代码已被注入服务器，已影响到所有用户，问题相当严重；若没有弹出此对话框，则说明只影响到恶意用户本身浏览器，问题不算严重。

例子

```
<iframe src=http://www.google.cn width=100 height=100></iframe>
```

单击“提交”按钮时，若Google的部分页面显现在当前页面，如图3-16所示，则说明存在XSS缺陷。

Offer



图 3-16 XSS 缺陷例 2

如果另一个用户从另一台浏览器访问此网页，在没有输入恶意代码情况下就能看见 Google 页面，则说明恶意代码已被注入服务器，已影响到所有用户，问题相当严重；若没有弹出此对话框，则说明只影响到恶意用户本身浏览器，问题不算严重。

面试试题 2

什么是 SQL 注入？

解答

SQL 注入属于命令注入，它是指发生于应用程序之数据库层的安全漏洞。简而言之，是在输入的数据字符串之中夹带 SQL 指令，在设计不良的程序中忽略了检查，那么这些夹带进去的指令就会被数据库服务器误认为是正常的 SQL 指令而运行，因此导致破坏。另外，SQL 还有一个变种是针对存储过程的。此时存储过程会接收一个参数，如果不对其进行检查就执行这个参数或者利用这些参数进行字符串连接，然后执行连接的结果，就可能会出现安全性问题。

分析

导致 SQL 注入的根源如下。

- SQL 命令可进行查询、插入、更新、删除等命令的串接，而以分号字符为不同命令的区别（原本的作用是用于 SubQuery 或作为查询、插入、更新、删除……等的条件式）。
- SQL 命令对于传入的字符串参数是用单引号字符所包起来。但连续两个单引号字符，在 SQL 数据库中，则视为字符串中的一个单引号字符。
- 在 SQL 命令中可以夹带注解（连续两个减号字符后的文字称为注解，或“/*”与“*/”所包起来的文字也称为注解）。

例子

假设某网站的登录验证的 SQL 查询代码为：

SELECT * FROM users WHERE (name = '" + userName + "') and (pw = '"+ passWord +"');

该网站的个人会员登录界面如图 3-17 所示。

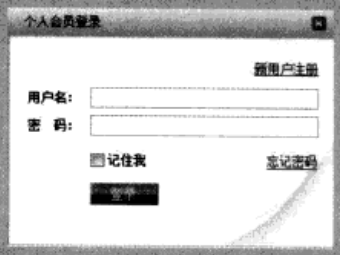


图 3-17 一个普通的用户登录界面

若在用户名文本框中输入:

' OR '1'='1

在密码文本框中输入:

' OR '1'='1

此时，将导致原本的 SQL 字符串被输入:

SELECT * FROM users WHERE (name = '' OR '1'='1') and (pw = '' OR '1'='1');

也就是说，实际上运行的 SQL 命令会变成:

SELECT * FROM users;

此时，恶意用户虽然没有账号密码，亦可成功登录网站。

关于测试用户登录的 SQL 注入问题的测试用例，可参考本书第 3.17.11 小节的相关内容。

面试试题 3

在你所测试的这个 B-S 结构软件中，你们在用户登录页面有何安全措施？

解答

（1）对用户名和密码数据进行了加密，在网上传输时不是明文。

（2）对登录时调用的 SQL 语句做了处理，防止 SQL 注入。

（3）要防止 URL 的 SQL 注入。

（4）用户名和密码的输入文本框都限制了输入长度，防止缓冲区溢出。

（5）如果 URL 输入错误（例如在后面加个单引号），则错误提示信息中不包含源代码内容等安全信息。

（6）处理了代码，防止 XSS 攻击。

（7）强制用户不能使用弱口令，例如长度要求在 6 到 15 位之间。

（8）当用户输错密码达到 5 次，该账户将被锁定（系统将会发解锁邮件到注册账户）。

（9）由于在该系统中用户名是电子邮件格式，所以要求输入必须符合国际规范，如 RFC822。

（10）在“忘记密码”的处理页面中也要防止 SQL 注入和 XSS 攻击。

（11）在“忘记密码”的所有输入文本框中都限制了输入长度，防止缓冲区溢出。

（12）如果“忘记密码”的处理页面的 URL 输入错误（例如在后面加个单引号），则错误提示信息中不包含源代码内容等安全信息。

（13）如果没有成功登录，则即使用户输入此软件其他页面的 URL，也会被重定向到登录页面。

（14）要注意重定向问题(若将客户端发过来的东西不加检查的当做 redirect 的参数，则很有

Offer

可能会出现问题)。

面试试题4

什么是Cookie欺骗?

解答

Cookie是指一些网站服务器存于用户计算机上的一个文本文件，为了辨别用户身份而储存在用户本地终端上（数据通常经过加密），它与服务器有一定的互交性，常存储用户名甚至网站登录密码。由于Cookie是保存在客户端浏览器中的，对通信协议无影响，攻击者可能通过伪造Cookie信息，绕过网站程序的验证，也不需要输入密码，以任意用户（如管理员）的身份登录网站。

进行Cookie欺骗可以有多种途径。

- 跳过浏览器，直接对通信数据进行改写。
- 修改浏览器，让浏览器从本地可以读写任意域名Cookie。
- 使用签名脚本，让浏览器从本地可以读写任意域名Cookie。
- 欺骗浏览器，让浏览器获得假的域名。

面试试题5

什么是网络钓鱼?

解答

在计算机安全领域中，钓鱼式攻击（Phishing，与钓鱼的英语fishing发音一样，又名“网钓法”或“网络网钓”）是一种企图从电子通信中，通过伪装成信誉卓著的法人媒体以获得如用户名、密码和信用卡明细等个人敏感信息的犯罪诈骗过程。这些通信都声称（自己）来自于风行的社交网站（如YouTube、Facebook、MySpace）、拍卖网站（如eBay）、网络银行、电子支付网站（如PayPal）或网络管理者（如雅虎、互联网服务供应商、公司机关），以此来诱骗受害人的轻信。网钓通常是通过E-mail或即时通信进行。它常常引导用户到URL与接口外观与真正网站几无二致的假冒网站输入个人数据，即使使用强式加密的SSL服务器认证，要想侦测网站是否仿冒实际上仍很困难。

网钓是一种利用社会工程技术来愚弄用户的实例。它凭借的是现行网络安全技术的低亲和度。种种对抗日渐增多网钓案例的尝试涵盖立法层面、用户培训层面、宣传层面与技术保全措施层面。

网钓技术最早于1987年问世，而首度使用“网钓”这个术语是在1996年。网钓技术主要有。

- 链接操控。
- 过滤器规避。



- 网站伪造。

3.15 兼容性测试

软件兼容性测试是指待测软件与硬件之间、与其他软件（包括操作系统、应用软件等）之间、软件本身新旧版本之间以及在不同的网络环境中是否存在冲突的测试。另外，还包括数据共享的兼容性测试。

例子

软件要在笔记本电脑上进行测试，经验证明，台式机上运行没有问题的软件，有可能在笔记本电脑上出现问题。

我们希望软件与软件、软件与硬件之间都可以“和谐共处”。

兼容性测试必不可少。随着软件开发的复杂性、平台及应用软件的多样化，另外还有软件实现方法的多样化，软件兼容性问题越来越多。

下面列出5个常见面试笔试题，以资读者参考。

面试试题1

什么是向前兼容？
什么是向后兼容？

解答

向前兼容处理的是一个以较新版本编译的程序集是否可以在较老版本上执行的问题。

向后兼容处理的是一个以较早版本编译的程序集是否可以在较新版本上执行的问题。

例子

例如使用日历软件2.0版本生成一篇带插图的文档,如果这篇文档能在1.0版本上被正确读取，这就是向前兼容；如果这篇文档也能在3.0版本上被正确读取，这就是向后兼容。

面试试题2

测试部门将对新软件进行兼容性测试。请问，欲测试此软件是否与其他软件相冲突，其方法和原则是什么？

解答

- 建立干净的测试环境，只保留最基本的软件，去除多余因素的干扰。



- 每次只把另一种软件和待测软件同时使用，便于定位兼容性问题。
- 选择最流行的应用软件、操作系统来进行兼容性测试。
- 对兼容软件进行分类，如画图类、杀毒类、财务类和数据库类等。每类软件选择几种。
- 对兼容软件选择不同生产厂商的软件。
- 对兼容软件尽量选择新的版本。

面试试题3

请问你在工作中有没有做过兼容性测试？怎么测试？

解答

做过。

笔者曾在汤森路透公司媒体部门工作，主要工作是测试公司的金融网站。我们需要进行跨浏览器测试。同时选择了多种浏览器，包括 Firefox2、Firefox3、IE6、IE7、IE8、Safari4、Maxthon10、腾讯浏览器。之所以选择它们，是因为考察了访问金融网站的流量数据。我们选择了所有用户比例大于 5%的浏览器（没有 Google 公司的 Chrome）。其中 IE6 是用户最多的，占 60%以上。

根据统计数据，我们重点是测试 IE6、IE7、Firefox3 这 3 种浏览器。

同时我们还选择了 Linux Fedora 9、Windows XP、Windows Vista、Mac（专用于 Safari4）等几种操作系统与不同浏览器构成测试矩阵。其中 Linux Fedora 9 与 Windows XP 是重点，原因同样是用户较多。

在跨浏览器测试中的测试方法是：在测试矩阵中的每个浏览器和操作系统组合上执行测试用例，验证软件产品的主要功能。在验证过程中，我们可能会发现一些界面和功能上的跨浏览器方面的软件缺陷。

接着从基本测试用例中选择一部分测试用例，在 IE 6、IE 7、Firefox 3 这 3 种浏览器上进行测试。再从这部分测试用例中选出一部分主要的，大约占 1/2 左右，在剩余的几种浏览器上进行测试。

另外，我们部门还有一个小组，负责手机的浏览器访问网站的测试。

面试试题4

请举出一个日常生活中软件不兼容的例子。

解答

例如微软的 Office 系列软件中的 Word 2007 生成的是.docx 文件。这种文件不能被 Word 2003 打开，尽管 Word 2003 生成的.doc 文件能被 Word 2007 打开。这就属于向前兼容问题。

面试试题5

什么是数据兼容测试？



解答

对于软件共享的数据，几个软件之间无须复杂的转换，即能方便地共享相互间的数据。

例子

例如微软 Excel 文件中生成的表格，可以剪切或复制后插入 Word 文件。

3.16 易用性测试

易用性（Usability）是一种以使用者为中心的设计概念，易用性设计的重点在于让产品的设计能够符合使用者的习惯与需求。易用性是交互的适应性、功能性和有效性的集中体现，其包含易理解性、易学习性和易操作性等，即易用性是指在指定条件下使用时，软件产品被理解、学习、使用和吸引用户的能力。

例子

对于家用热水器的两个水龙头，大多数人不加思考就知道标记为红的是热水，蓝的是冷水。如果我们设计得与此相反，就会造成用户的接受及使用的不便。

例子

一般情况下，安装卸载时在屏幕上都会有进程条显示，这样用户就能够了解大致的进度；否则用户在不了解进度的等待中将会感觉不舒适。

例子

在汤森路透的金融网站上，用户可订阅一年或半年的商业报告，但是退订规则包括退款方式等并没有告知用户。笔者提出了这个问题，最终此问题被解决，页面上加了链接来详细说明退订规则。

易用性主要包括。

- 易理解性。
- 易学习性。
- 易操作性。
- 吸引性。
- 依从性。

易用性和人体工程学密切相关，人体工程学的主要目标是达到易用性。

另外，还需要考虑残疾人的使用。这称为辅助选项测试（Accessibility Test），也就是为残障人士使用而进行的测试。这里所说的残疾包括许多种，如视力损伤、听力损伤、运动损伤、认知和语言障碍等。

Offer!

易用性测试带有一定的模糊性和主观性。

在实际测试中，易用性测试主要分为以下 4 个方面。

- 安装/卸载测试。
- 功能易用性测试。
- 界面测试。
- 辅助系统测试。

在进行设计时，我们也需要参考一下同行业的流行软件。由于先入为主的关系，人们很可能已经接受了已存在软件的界面和功能设计。

下面来讨论几道基本的面试题。

面试试题 1

易用性和正常功能之间是否存在冲突？

解答

有可能会存在冲突。由于易用性带有一定的主观性，不同的人会有不同的易用性体会，例如有的人看文档喜欢白色的背景，有的人则喜欢黄色的。有时两个易用性要求本身就是相冲突的。

例子

例如在要求数据库查询快的同时，增删记录也快。为了查询快而增加了索引，这样就会造成增删变慢。

在易用性的实现时常会牺牲或影响一些正常的功能，而且新加入的代码会引入新的 Bug。

面试试题 2

怎样保证网站的易用性？

解答

首先，我们需要进行调查。

- 网站用户是谁？
- 网站用户具备哪些知识？
- 网站用户想做什么？
- 用户使用网站时，场景是什么？
- 哪部分任务可以交给程序办理？哪些需要用户来做？

其次，我们还要重视以下几点。

- 易理解：用户在几秒钟内就可观察得知网站是做什么的，以及知道可以采取的行动。
- 易操作：当初次接触网站功能或界面时，用户就能很容易实现这些功能，最好不用帮助

Offer

文档就能使用网站。

- 易学习：通过简单的帮助文件，用户就可解决所遇问题。
- 效率性：当用户熟悉网站的功能或界面后，完成任务的速度应当很迅速。
- 出错率：用户在网站的使用过程中出现的错误较少，且这些错误不严重。用户及软件系统从错误中恢复也较容易。
- 满意度：用户对设计满意度较高。

面试试题3

软件界面的易用性要求是什么？

解答

- 符合行业标准和规范。
- 直观易懂。
- 一致性：即与市场上同类流行软件应保持某些重要的风格或设计一致；或作为一套软件之一与系列内其他软件保持风格或设计一致。
- 灵活性。用户可以采取多种方式进行同样目的的操作。

例子

例如在微软公司的 Word 软件中，复制操作可以按 Ctrl+V 组合键，也可在右键菜单中选择“复制”命令。

- 舒适性，如用法舒适、视觉舒适、反馈舒适。
- 正确性。
- 实用性。

面试试题4

针对“帮助”功能的易用性测试，请写出 5 条测试用例。

解答

（1）“帮助”文档中的性能介绍与说明要与软件实际性能配套一致。

（2）在打包新系统时，对做修改的地方在“帮助”文档中要做相应的修改。

（3）操作时要提供及时调用“帮助”的功能，常用 F1 键。

（4）在界面上调用“帮助”时应该能够及时定位到与该操作相对应的“帮助”位置。也就是说，“帮助”要有即时针对性。

（5）最好提供目前流行的联机帮助格式或 HTML 帮助格式。

（6）用户可用关键词在“帮助”索引中搜索所要的帮助，软件也应提供“帮助”主题词。

Offer

（7）如果没有提供书面的帮助文档，则最好有打印“帮助”的功能。

（8）在“帮助”中应提供产品技术支持的联系方式。

面试试题 5

易用性软件缺陷处理的流程是怎样的？

解答

通常，易用性软件缺陷首先被提交到缺陷跟踪系统上，再经过软件缺陷分配部门的审核后，然后它会被分配到设计部门，而不是开发部门。如果设计部门同意这是个缺陷，则设计部门会修改设计文档并通知开发部门和测试部门。开发部门据此修改概要设计及详细设计文档，测试部门据此修改测试用例。之后这个软件缺陷将被开发部门解决，解决后开发人员在缺陷跟踪上更新这个缺陷报告状态为“已解决”，最后测试部门在新版本进行返测，确认缺陷解决后关闭这个缺陷报告。

如果设计部门不同意这是个缺陷，则测试人员将比较被动，除非测试人员有非常充足的理由来支持自己的观点。

说明

在有关设计的缺陷认定上，通常设计部门拥有最高决定权。

例子

笔者在摩托罗拉公司测试手机时曾发现一个问题：当删除短信时，手机没有要求用户再次确认删除就直接删掉短信。笔者认为这样会很容易发生误删，经过多次和设计部门的拉锯战，笔者举证了 NOKIA 等多家手机公司的操作方式，最终说服了设计部门。

通常易用性软件缺陷的严重级都比较低，时常被归类为建议，还经常被拒绝掉不进行修改。有时会被修改为 Won't Fix 状态，即承认是错误但不修改。

说明

开发人员通常不喜欢易用性软件缺陷的提出。他们往往觉得解决这种缺陷没有太大意义，而往往希望解决技术上的难题而不是由于商业原因而产生的毛病。

3.17 测试用例场景模型 12 例

在面试中，时常有这样一种考题。面试官画出一个软件界面或指明一种物品，要求应聘者写出测试用例。

记得好几年前去微软应聘，面试官问我如何测试圆珠笔，要求我给出测试用例。还有一道题是对冰箱的测试给出测试用例。

Offer!

这种考题基本在100%的软件测试工程师面试中都会出现，当然大部分都是对一个软件界面写出测试用例。

这种题目考查的要点如果不是软件界面，如圆珠笔，那么考查的就是发散思维和逻辑性。此时应聘者应力争想到方方面面可能发生的缺陷，不要遗漏一些重要的环节。面试官通常都会问设计思路，正确的回答是先分成大的方面，再逐一细分。

如果是给出软件界面的面试题，那么就是考查软件测试基本理论和项目经验。面对一个模块，如何使覆盖率尽量达到100%，如何使你的回答超出对方的要求，见人所未见，不深入钻研软件测试理论，不经常加以总结前人和自己的经验，是很难做到的。

本节是我及前人进行的一些心血的总结，对12个典型场景设计了测试用例，在这里与大家分享一下，读者可将之立即用于实践。当然，实际测试中的场景很多，远不止这12个，我在这里抛砖引玉，以供读者参考。

事实上，在很多软件测试中都会遇到一些类似的场景，所以大家平日应做好测试用例的总结整理工作并使之不断完善，这样不但利于复用，而且提升了工作质量。

3.17.1 边界值测试

在值域的边界值处，Bug经常会产生。下面是一些测试用例。

- 对符合条件的值域内各个端点值进行测试。
- 测试值域的各个端点值+1。
- 测试值域的各个端点值-1。
- 对一个远远小于值域最小值的值进行测试。
- 对一个远远大于值域最大值的值进行测试。

例子

例如用户需求是在一个文本框中只能输入500到2000中的一个数字，那么应如何来进行边界值测试呢？

- 根据上面用例1，我们要测试500和2000这两个值。
- 根据用例2，我们要测试501和2001。
- 根据用例3，499和1999也需要被测试。
- 根据最后两条测试用例，有很多值可选，我们选10和10000来测试。

3.17.2 安装和卸载测试

对新开发的软件的安装和卸载能力是必须被测试的环节。下面是一些测试用例。

- 新开发软件不应与现存的或市场上主流的软件冲突。
- 软件授权的有效期需要被测试。
- 安装过程中所有的选项应完整无遗漏。

Offer

- 软件的新版本不应与旧版本冲突。
- 当一个新版本已被安装时，旧版本不应允许被安装。
- 卸载软件时，此软件的相关文件都应被删除。
- 在安装或卸载过程中关闭电源，当打开电源后再次安装或卸载，仍旧一切正常。.
- 卸载不应影响其他软件使用。
- 不应出现界面 Bugs。
- 在安装前，安装程序需要检查磁盘空间大小是否符合安装要求。
- 安装开始时，安装程序需要提醒用户所有的前提条件。
- 安装和卸载过程中都应有进程条显示，显示进度应与实际进度一致。
- 安装和卸载步骤不应过于烦琐。
- 在安装和卸载过程中，重启系统次数不应过多。
- 安装程序不应把软件安装在系统中的偏僻角落，以致于用户无法找到。
- 卸载程序应当被提供。
- 运行软件一段时间后，卸载软件，卸载应当是成功的，相关文件也会被删除。

3.17.3 表单测试

- 用户名不应为空。
- 电子邮件地址不应为空，并且“@” 和“.”应存在。
- URL 不应为空，“.”应存在。
- 所有必填项都必须有值。
- 提交和取消键功能正常。
- 在所有文本域检查非法字符。
- 在一组单选框中，同一时间仅有一个单选项被选中。
- （F）正则表达式检查。
- 边界值检查。
- 按下 Tab 键可移动焦点，遵循从上到下、从左到右的顺序。
- 关联项检查。

例子

有城市和国家两栏，北京对应中国，如果将中国改为美国。提交时系统应当提示错误。

（J）检查脚本注入漏洞

3.17.4 链接测试

- 所有的链接应指向正确页面。
- 如果被链接的页面不存在，则友好的提示页面应显示，告知用户。

Offer!

- 孤立页面不应当存在。
- 每个超链接的文字应当简洁。
- 外部链接应当有效。

3.17.5 界面集成测试

- 需参考一些同类的著名的流行软件界面，界面设计不可相差太远。
- 同一界面所有组件需要风格统一、有序、整洁。
- 各个界面风格统一。
- 颜色运用适当，不可散乱。
- 字体及格式应当适当。
- 提示信息应当存在。
- 对所有可选列表，默认选项应当存在。
- 默认的焦点应当存在。
- 对表单项，默认值应当存在。
- 功能实现须符合用户习惯。

3.17.6 窗口界面测试

- 打开窗口的大小、位置应与上次打开时相同。
- 当我们关闭窗口时，如果软件没有设置自动保存，而我们又没有手动保存的话，则提示信息应当弹出询问是否保存信息。
- 如果窗口可以移动，那么此窗口不能被移动到一个不能移回的位置。
- 当窗口尺寸变化时，所有组件的相对位置及大小应当相应变化。
- 对于一些窗口（如登录框），最大化和最小化按钮不应当存在。

3.17.7 菜单界面测试

- 对于每个菜单项，字符只能占一行，而不能占两行。
- 如果一个菜单项按下会弹出一个对话框，则此菜单项的字符应以“...”结尾。
- 如果一个菜单项按下会弹出一个子菜单，则此菜单项的字符应以“->”结尾。
- 提倡浅菜单，如果菜单项过多，应分置于多个菜单。
- 在一个菜单中，功能相近的菜单项应当分组，组之间应用分隔线进行分隔。
- 重要的菜单项应当放在顶端，重要的菜单应当置于最靠前的位置。
- 通常，“文件”应当是第一个菜单，“帮助”应当是最后一个菜单。

3.17.8 单选框/复选框测试

- 不可使用的单选框/复选框应被置灰。

Offer

- 一组单选框可包含 2~7 个单选框，如果超过 7，则建议使用列表。
- 单选框/复选框后面使用的文本应尽量简洁。
- 两个单选框/复选框后面使用的文本不应当过于相似而引起混淆。

3.17.9 超长字串 Bug

这是一个非常常见的问题。有时超长字串会造成缓冲区溢出，但不是所有的超长字串 Bug 都是缓冲区溢出。

[测试步骤]

1. 输入超长字串（例如 10000 个字符）到一个可输入域。
2. 按下“保存”或“提交”或“OK”等确认键。

[正确结果 1]

“保存”或“提交”或“OK”等确认活动失败，错误信息提示字串超长。

[正确结果 2]

文本域存在输入字符数量限制，超过一定数量，用户不可以继续输入。

[存在缺陷的结果 1]

“保存”或“提交”或“OK”等确认活动非常缓慢。

[存在缺陷的结果 2]

“保存”或“提交”或“OK”等确认活动结束后，名字中仅有部分字符被显示。

[存在缺陷的结果 3]

“保存”或“提交”或“OK”等确认活动结束后，某些功能不再正常，如程序崩溃、页面消失等。

3.17.10 新建-编辑接口测试

在软件开发中，很多相联系的功能是由不同的开发小组开发的，由于信息的不能充分共享，因此造成功能接口处常常出现问题。即使是同一个人开发两个模块，也容易对相联系功能的检查有所忽略。因此，功能接口处往往存在软件缺陷，需要被测试人员重点关注。

“新建”和“编辑”是大多数软件都具有的功能，它们的功能很相像，很多地方重复，因此很容易发生接口问题。对它们的测试，是个典型的功能接口测试。

在实践中可发现，在“新建”模块中，通常对用户输入的检查比较严格，而在“编辑”模块中往往没有那么严格，因此 80%的接口缺陷都发生在“编辑”模块。这可能是由于开发人员首先实现的是“新建”模块。由于“新建”模块的大量代码可以复用于“编辑”模块，因此开发人员在实现“编辑”模块时，心理上往往有所懈怠。

我们在测试时，可以运用以下测试用例。

- 将所有应用于“新建”模块的测试尽可能地应用于“编辑”模块。尤其要注意“编辑”模块的所有输入项的默认值。

Offer!

- 针对“新建”与“编辑”的不同点进行测试。
- 在“编辑”模块，首先为各项输入正确值，提交后，将前面某项或几项改为错误值，再次提交。有可能发现提示信息漏掉了几项，甚至有可能使此部分程序崩溃。
- 更改URL，在正确的URL后面加单引号或者试图注入代码。
- 能否“编辑”两个“新建”项，使一些关键信息重复？
- 当“新建”项数目已经达到了最大值，能否通过编辑“新建”项，使数目超过最大值（数据结构溢出）？

3.17.11 用户登录的SQL注入测试

在输入用户名和密码进行登录时，SQL注入是个常见的安全漏洞。黑客无须知道密码，甚至无须知道用户名，就可能登录成功。

下面是一些测试用例。

- 输入正确的用户名，在密码域中输入：' or 1=1—，提交。
- 输入正确的用户名，在密码域中输入：" or 1=1--，提交。
- 输入正确的用户名，在密码域中输入：or 1=1--，提交。
- 输入正确的用户名，在密码域中输入：' or 'a'='a，提交。
- 输入正确的用户名，在密码域中输入：" or "a"="a，提交。
- 输入正确的用户名，在密码域中输入：') or ('a'='a，提交。
- 输入正确的用户名，在密码域中输入：' or '1=1，提交。
- 输入正确的用户名，在密码域中输入：" or "1=1，提交。
- 输入正确的用户名，在密码域中输入：' or 1=1-，提交。
- 输入正确的用户名，在密码域中输入：" or 1=1-，提交。
- 输入正确的用户名，在密码域中输入：") or ("a"="a，提交。
- 保持密码域为空，在用户名中输入：'or'='or'，提交。
- 保持用户名为空，在密码域中输入：'or'='or'，提交。

3.17.12 SQL查询语句测试

这种测试实际上属于白盒测试。很多软件需要用到数据库，因此代码中常常会包含SQL查询。在进行SQL查询语句测试时，要注意以下一些方面。

- 注意边界值，包括between-and、in、<、≤、>、≥的应用。
- 注意相似字段的混淆。例如terminated_date被误用为end_date。
- 注意别名起的和某字段重名时，在group by等子句中，别名前要加上表名，否则数据库可能因分不清而得不到结果。
- 注意表的选取，不要误用。
- 注意有些字段已被代替或废弃。



- 对比需求，限制条件不可缺少。例如有时 and 或 or 语句会被漏掉几条。
- 注意对 0 值和 NULL 值的处理。
- 注意对用户输入数据的格式处理。

3.17.13 面试题

面试试题 1

以上列举了 12 个测试模型，下面给出一道面试真题，如图 3-18 所示。

请对以下软件界面写出测试用例：

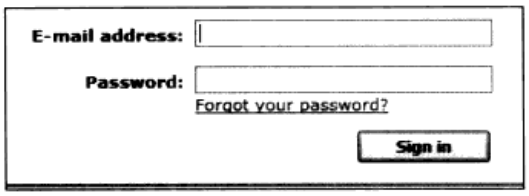


图 3-18 某网站登录对话框

解答

- 组件相对大小和位置有序、协调、整齐。
- 每组组件的字体、风格保持一致。
- 输入密码时，密码显示不能是明文，应以星号或其他符号代替。
- 如果该窗口可以移动，那么当移动该窗口时，所有组件都将随之移动，并且彼此间的相对位置保持不变。
- 如果该窗口可以移动，那么不能移动到一个移不回来的位置。
- 随着字符的不断输入，邮箱地址和密码域文本框不应随之拉长，应保持不变。
- 最大化、最小化按钮不应使用。
- 对话框右上角应设置“关闭”按钮，或在“Sign in”按钮旁边设置“取消”按钮。
- 按下 Alt+F4 键，该窗口可以被关闭。
- 按下 Tab 键，焦点移动顺序应是从上到下，从左到右。按下 Shift+Tab 组合键，则与之相反。
- 对邮箱地址和密码域进行超长字串测试。
- 提交时，邮箱地址和密码域不能为空。
- 如果输入正确，则单击“Sign in”按钮后可以正常登录。
- 如果输入不正确，则单击“Sign in”按钮后应有友好而足够的信息提示用户。
- 参照需求，邮箱地址中包含中文时（也为有效邮箱地址）是否也可以登录？
- 参照需求，是否同一用户可以在多台机器上同时登录？需要进行测试。



- 这是个非限制性对话框，那么不可以对对话框以外页面进行操作。焦点不能移动到对话框之外的页面。
- 在IE浏览器上按下回车键，对应的是单击“Sign in”按钮。
- 对邮箱地址和密码域进行SQL注入测试。
- 对“Forgot your password?”超链接进行键接测试。
- 单击“Forgot your password?”超链接，按照提示一步步操作，如果是已注册用户，则找回密码的功能可以实现；如果不是已注册用户，则不可以得到密码。
- 单击“Sign in”按钮后观察URL，密码不应以明文的形式出现在URL中。
- 对此登录页面使用性能自动化工具如LoadRunner或Jmeter进行性能测试，仿真大量不同用户同时登录，进而找出数据库或网络瓶颈，或找出代码可以优化之处。
- 跨浏览器测试。在不同的流行的浏览器上测试功能和界面。
- 查看源代码，对SQL查询语句进行测试。
- 用安全测试工具如微软公司的Fortify对源代码进行安全测试。

思索

你能想出更多的测试用例吗？例如针对URL？开动脑筋吧，你一定行。

Offer!

第4章 软件测试基础2——计算机应用

4.1 概　述

软件测试是计算机学科的一个分支，也是一门计算机综合学科。除了软件测试基本概念和原理之外，相关的计算机领域的知识也必须掌握。

例如测试一个 B/S 结构的软件，该软件使用 Java 语言在 Linux 平台上开发，采用了 Oracle 数据库，使用了 Apache 和 Tomcat 服务器。

那么，作为一个测试工程师，我们必须具有 Java、Linux、数据库及网络方面的知识，才有可能设计并进行好这个软件的测试。仅仅知道什么是集成测试，什么是回归测试，也许从朋友处抄到了一些黑盒设计用例场景模型，是不可能做好软件测试的。

在各个企业对测试工程师的招聘要求上一定会列出一些对计算机相关领域的要求。

例子

请看 Google 公司对某个测试工程师的招聘需求（节选）：

- Good programming skills in at least one of the following languages: C/C++, Java or Python.（熟练掌握 C/C++、Java、Python 语言中至少一种）
- Experience working on UNIX/Linux.（有 UNIX/Linux 方面的工作经验）
- Excellent shell scripting skills (awk, sed).（精通 UNIX Shell 中的 awk 及 sed 编程）
- Good scripting skills in at least one of the following scripting languages: perl, vbscript, php.（熟练掌握 perl、vbscript、php 这几种脚本语言中的至少一种）
- Excellent knowledge of Internet/Web technologies, such as web browsers, http, and html.（具有深厚的网络技术知识，如 Web 浏览器、http、html）

在本书中，我们将软件测试相关的计算机应用、软件测试概念与原理并列，作为软件测试理论基础的两大部分。



作为一名出色的软件测试工程师，对于某一个相关计算机领域不如开发人员知识深入没有关系，但是知识面一定要广，必须对多个领域有相当程度的了解。

思索

当然，深且广是更好的，不过一个人的精力有限，我们也反对为了工作完全放弃生活的人生态度。这里应当以80-20原则来要求自己，将80%的精力投入到最有用的领域。

4.1.1 难为无米之炊

通常来说，任何行业都是需要专业背景的，跨行业当然也是可能的，但是前提是必须投入大力气钻研新行业。在软件测试领域，虽然门槛也许不高，甚至有些非工科专业背景的人士都转做这行，但是仅凭一些测试基本概念和原理，他们只能从事最没有技术含量的工作，当然薪水也不会太高，在单位里也不会受到重视，升迁无望、加薪缓慢，而且时间越久越没前途。显而易见，随着正规院校软件测试专业的设置，越来越多科班人士的进入，这样缺乏专业技术背景的人将首先被淘汰。

因此，如果不是计算机专业出身或非工科背景，如果希望从事软件测试工作，一定要注意苦修这些计算机领域的知识。

思索

实际上，再往深一层次讲，我们也应当学习计算机专业的基础课，如离散数学、数据结构等。这些都是专业的基础，基础夯实，受用无穷；基础不牢，欲速则不达。

4.1.2 计算机应用领域汇总

在本章中指出了目前市场上软件测试工程师需要掌握的一些计算机应用领域，对每个领域进行了简单介绍，指出了面试中的考点和重点。我们对每个领域给出了5道典型的面试题，并且加以解析。

这些章节分布如下

4.2 C++语言知识考查。

4.3 C语言知识考查。

4.4 Java语言知识考查。

4.5 UNIX Shell知识考查。

4.6 网络知识考查。

4.7 质量管理知识考查。

4.8 Linux知识考查。

4.9 VB/VBScript/JavaScript语言知识考查。

4.10 HTML/XML知识考查。



4.11 Win32 开发知识考查。

4.12 C#语言知识考查。

4.13 数据库知识考查。

4.14 主流测试工具知识考查。

除此之外，还有一些与测试相关的计算机应用领域，例如 Ruby 语言，限于篇幅，此处就不再进行介绍。

4.2 C++语言知识考查

C++程序设计语言是在 C 语言的基础上发展而来的优秀的面向对象语言。C++语言是由美国 AT&T 贝尔实验室的本贾尼·斯特劳斯特卢普（Bjarne Stroustrup）博士在 20 世纪 80 年代初期发明并实现的。C++是一种静态数据类型检查的、支持多重编程范式的通用程序设计语言。它支持过程化程序设计、数据抽象、面向对象程序设计、制作图标等泛型程序设计等多种程序设计风格。

C++程序设计语言由以下 4 个“子语言”组成。

- C 子语言。C++支持 C 语言的几乎全部功能，在语法上与 C 语言仅有极细微的差别。
- 面向对象的 C++。C++作为一门面向对象的语言而闻名，该特点在这里不再详述。
- 泛型编程语言。C++强大（但容易失控）的模板功能使它能在编译期完成许多工作，从而大大提高运行期效率。
- STL（C++标准模板库）。随着 STL 的不断发展，它已经逐渐成为 C++程序设计中不可或缺的部分，其效率可能比一般的 naive 代码低些，但是其安全性与规范性使它大受欢迎。

C++程序设计的语言特性。

- 预处理器：C++主要有 3 个编译阶段：预处理、转译成目标代码和链接。在预处理阶段，会将预处理器指令替换成源代码，然后送到下一个编译阶段。
- 预处理器指令和宏：预处理指令的运作方式是根据用户定义的规则，简单地把记号字符序列置换成其他的记号字符序列。
- 模板：模板（Template）指 C++编程语言中的函数模板（function template）与类模板（class template），这种观念是取材自 Simula 的泛型程序设计。C++之父 Bjarne 认为可以通过 C++的重载（overloading）功能来表达泛型。
- 对象：是指数据和处理数据指令的集合，它有 3 种特性：状态、行为和同一性，并使用消息交互。而类定义了对象的特质，包括对象的属性和行为。C++以数据成员表达属性，以成员函数表达行为。
- 封装：是指将数据和处理数据的过程组合起来，仅对外提供接口，从而实现信息隐藏的功能。



- 继承：是指子类继承父类，获取父类的除私有特质外的全部特质，包括属性和方法，以达到代码的重用。C++同时支持公有继承和私有继承。
- 多态：一般的多态是指动态多态，即使用继承和动态绑定来实现。动态绑定就是在代码运行时再决定到底如何执行代码。C++的多态分为静态多态和动态多态。需要注意的是，动态多态必须结合继承和动态绑定方式实现。

4.2.1 常见考点汇总

- 头文件的结构和作用。
- 命名规则。
- 运算符的优先级。
- 循环语句的效率问题。
- 常量的使用。
- 常量定义规则。
- 参数的规则。
- 引用与指针的比较。
- 内存分配方式。
- 指针和数组的对比。
- 如何杜绝“野指针”。
- 指针参数如何传递内存。
- Malloc()/Free()使用要点。
- New()/Free()使用要点。
- 成员函数的重载、覆盖和隐藏。
- 运算符重载。
- 构造函数和析构函数。
- 拷贝构造函数和赋值函数。
- 类的继承和组合。

4.2.2 经典考题及解析

面试试题1

C++程序的内存通常如何分配？

解答

要把本试题回答得全面，不仅要对内存分配有充分的认识，同时还要对C++程序中变量、函数等的分类有一定的认识。该题目属于考察面试者的基本功。



- 全局数据区(data area)：通常存储全局变量、静态数据和常量。
- 代码区(code area)：所有类成员函数和非成员函数代码。
- 栈区(stack area)：在执行函数时分配的局部变量、函数参数、返回数据及返回地址等都是在栈区上创建的。
- 堆区(heap area)：从堆上分配的内存是在程序运行时由程序员申请的内存，在程序结束时由程序员负责释放。

面试试题2

请谈一下你对C++单元测试框架CppUnit的认识？并举例说明。

解答

C++单元测试框架CppUnit的认识可以从两方面回答,一是说明CppUnit是用来做什么的，二是可以列出该框架的特点。

CppUnit是什么?

CppUnit是C++单元测试框架，是个开源项目，最初版本移植自Junit。其主要功能就是对单元测试进行管理，并可进行自动化测试。

CppUnit的特点?

- XML形式输出结果，可以获取附加数据以便于调试程序。
- 类似编译器的文本输出，并与IDE集成。
- 帮助宏更有利于测试套件的定义，并提供分层测试夹具的支持。
- 测试注册以减少重新编译的需要。
- 测试插件可以更快的编译测试。
- 可以保护封装测试的执行。
- 可支持Mfc 和QT 测试运行。

CppUnit实例分析

```
class ComplexNumberTest : public CppUnit::TestCase {
  public:
    ComplexNumberTest( std::string name ) : CppUnit::TestCase( name ) {}
    Void runTest(){
        CPPUNIT_ASSERT( Complex (10, 1) == Complex (10, 1) );
        CPPUNIT_ASSERT( !(Complex (1, 1) == Complex (2, 2)) );
    }
}
```

这是一个非常简单的例子,但却可以很好地解释CppUnit。由CppUnit派生出的子类TestCase覆盖了方法 runTest()，当想检查一个值是否正确时,只需要调用 CPPUNIT_ASSER(bool)断言方法，参数即为所要测试的方法，如果测试用例通过，会返回true，否则返回false。



面试试题3

请分别详细解释成员函数的覆盖、重载和隐藏的区别和联系。

解答

想学好C++程序设计语言，必须要搞清楚这几个概念的区别和联系。

成员函数被重载的特征如下。

- 相同的范围（在同一个类中）。
- 函数名字相同。
- 参数不同。
- virtual关键字可有可无。

覆盖是指派生类函数覆盖基类函数，特征如下。

- 不同的范围（分别位于派生类与基类）。
- 函数名字相同。
- 参数相同。
- 基类函数必须有virtual关键字。

C++成员函数的“隐藏”是指派生类的函数屏蔽了与其同名的基类函数，规则如下：

- 如果派生类的函数与基类的函数同名，但是参数不同，则此时不论有无virtual关键字，基类的函数都将被隐藏（注意应避免与重载混淆）。
- 如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有virtual关键字，则此时基类的函数被隐藏（注意应避免与覆盖混淆）。

面试试题4

谈谈你对C++语言中虚函数的认识，并解释纯虚函数的用途。

解答

首先，C++通过虚函数实现多态。“无论发送消息的对象属于什么类，它们均发送具有同一形式的消息，对消息的处理方式可能随接收消息的对象而变”的处理方式被称为多态性。在某个基类上建立起来的类的层次构造中，可以对任何一个派生类的对象中的同名过程进行调用，而被调用的过程提供的处理可以随其所属的类而发生改变。虚函数首先是一种成员函数，它可以在该类的派生类中被重新定义并被赋予另外一种处理功能。

虚函数允许派生类取代基类所提供的实现。编译器确保当对象为派生类时，即派生类的实现总是被调用，即使对象是使用基类指针访问而不是派生类的指针。这样就允许基类的算法被派生类取代，即使用户不知道派生类的细节。

派生类可以完全取代基类成员函数（覆盖），也可以部分取代基类成员函数（增大），后者

Offer!

还可由派生类成员函数调用基类成员函数来完成。

相比而言，虚成员函数是动态确定的（在运行时）。也就是说，成员函数（在运行时）被动态地选择。该选择基于对象的类型，而不是指向该对象的指针/引用的类型。这被称为“动态绑定”。

纯虚函数是一个抽象类，是不能被实例化的，它用来规范派生类的行为，实际上也就是所谓的“接口”。

面试试题 5

（1）static 全局变量与普通全局变量有什么区别？
（2）static 局部变量和普通局部变量有什么区别？
（3）static 函数与普通函数有什么区别？

解答

（1）在全局变量（外部变量）的说明之前再冠以 static 就构成了静态的全局变量。全局变量本身就是静态存储方式，静态全局变量当然也是静态存储方式，二者在存储方式上并无不同。其主要区别在于非静态全局变量的作用域是整个源程序，当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的；而静态全局变量则限制了其作用域，即只在定义该变量的源文件内有效，而在同一源程序的其他源文件中不能使用。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用，因此可以避免在其他源文件中引起错误。static 全局变量只初使化一次，以防止在其他文件单元中被引用。

（2）将局部变量改成静态变量后是改变了它的存储方式，即改变了它的生存期。而将全局变量改成静态变量后则是改变了它的作用域，限制了它的使用范围。static 局部变量只被初始化一次，下一次依据上一次的结果值。

（3）static 函数与普通函数的作用域不同，仅在本文件。只在当前源文件中使用的函数应该说明为内部函数（static），内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，则应该在一个头文件中说明，若要使用这些函数的源文件就要包含这个头文件。static 函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝。

4.3 C 语言知识考查

维基百科中对 C 语言的介绍如下。

“C 语言，是一种通用的、过程式的编程语言，广泛用于系统与应用软件的开发，其具有高效、灵活、功能丰富、表达力强和较高的移植性等特点，在程序员中备受青睐。

C 语言是由 UNIX 的研制者丹尼斯·里奇（Dennis Ritchie）和肯·汤普逊（Ken Thompson）在 1970 年研制出的 B 语言的基础上发展和完善起来的。目前，C 语言编译器普遍存在于各种不

Offer!

同的操作系统中，例如 UNIX、MS-DOS、Microsoft Windows 及 Linux 等。C 语言的设计影响了许多后来的编程语言，例如 C++、Java、C#等。

后来为了避免各开发厂商用的 C 语言语法产生差异，由美国国家标准局（American National Standard Institution）为 C 语言制订了一套完整的国际标准语法，称为 ANSI C，作为 C 语言的标准。20 世纪 80 年代至今的有关程序开发工具，一般都支持符合 ANSI C 的语法。”

C 语言是一种结构化语言。它层次清晰，便于按模块化方式组织程序，易于调试和维护，其表现能力和处理能力极强。它不仅具有丰富的运算符和数据类型，便于实现各类复杂的数据结构，同时还可以直接访问内存的物理地址，进行位一级的操作。由于 C 语言实现了对硬件的编程操作，因此它集高级语言和低级语言的功能于一体，既可用于系统软件的开发，也适合于应用软件的开发。此外，C 语言还具有效率高、可移植性强等特点，因此被广泛地移植到了各类计算机上，形成了多种版本的 C 语言。目前最流行的 C 语言有以下几种。

- Microsoft C 或称 MS C。
- Borland Turbo C 或称 Turbo C。
- AT&T C。

上述 C 语言版本不仅实现了 ANSI C 标准，而且在此基础上各自进行了一些扩充，使之更加强大。

Linux 的内核几乎全是用 C 语言写成的（只是一些架构相关部分是用汇编语言写成的），这也说明 C 语言在嵌入式系统中也得到了广泛应用。

C 语言的常用编译器有。

Turbo C、VC++、Win TC、GCC、Dev C++。

C 语言是高等院校普遍开设的一门课程，是软件工作者必须掌握的一种工具。C 是 C++的基础，C++语言和 C 语言在很多方面是兼容的。在 C 语言的基础上学习 C++事半功倍。

正是由于 C 语言的普遍性和基础性，很多公司在招聘普通的黑盒测试工程师时，对 C 语言的考查更是重中之重。相应地，大部分测试工程师职业培训机构中也把 C 语言教学作为重点。

4.3.1 常见考点汇总

- C语言的典型特点，怎样运行。
- 预处理。
- 数据类型、基本数据类型的定义及使用、3 基本数据的表达方式和方法。
- 算术运算符、表达式（如逗号表达式）。
- 变量作用域。
- 关系运算符及其规律、运算的优先级别、逻辑运算符、条件运算符。
- 各种循环语句、循环嵌套、终止循环。
- 数组的定义及特点、数组与指针关系。
- 形参和实参。

Offer

- 二进制和十进制、八进制、十六进制之间的转换。
- 共用体和结构体。
- 函数的定义及调用（注意递归）。
- 一般算法（如冒泡排序法、折半查找法）。
- 常用库函数。
- 函数指针与指针函数。
- 文件的操作。
- 单向链表。

4.3.2 经典考题及解析

面试试题 1

请指出下面程序中的错误，并加以改正。

```
1    # include<stdio.m>
2    int p=2.
3    int main(int)
4    {
5    if(p=2);
6       printf("p is 2!");
7    or
8       printf("p is not 2!");
9    return 0;
10   }
```

解答

（1）第 1 行 stdio.m 应改为 stdio.h。

（2）第 2 行应当以分号结尾。

（3）第 3 行 int main(int)应改为 int main(void)或 int main()。

（4）第 5 行应去掉分号，因为语句并未结束。

（5）第 7 行 or 应改为 else。

面试试题 2

请写出以下程序的输出结果：

```
#include <stdio.h>

int a=2, b=8;

void change();
```

Offer!

```
int main(void)
{
    printf("\nBefore calling change(), a=%d, b=%d", a, b);
    change();
    printf("\nAfter calling change(), a=%d, b=%d", a, b);
}

void change(void)
{
    int a=50, b=60;
    printf("\nIn change(), a=%d, b=%d", a, b);
}
```

解答

程序输出结果如下：

```
Before calling change(), a=2, b=8
In chage(), a=50, b=60
After calling chage(), a=2, b=8
```

分析

本题主要考查局部变量和全局变量。局部变量是在函数中声明的，其独立于程序中的其他变量。例如，int a=2, b=8; 这一句定义的是全局变量，它不受change()函数中定义的局部变量（int a=50, b=60）的影响。

下面解释一下C语言中的局部变量和全局变量。

- 局部变量也称为内部变量，它是在函数内进行定义说明的。其作用域仅限于函数内，也就是说，若离开该函数后再使用这种变量则是非法的。
- 全局变量也称为外部变量，它是在函数外部定义的变量，其不属于哪一个函数，而是属于一个源程序文件。其作用域是整个源程序。在函数中使用全局变量，一般应进行全局变量说明。只有在函数内经过说明的全局变量才能使用。全局变量的说明符为 extern。在一个函数之前定义的全局变量，在该函数内使用可不再加以说明。

面试试题3

要使用bsearch()函数在包含300万个元素的数组中查找一个元素，最多需要进行多少次比较？

解答

21

分析

常用库函数 bsearch()位于头文件 stdlib.h 中，用于在数组中查找一个元素。它采用二分法。

Offer

数组中的元素必须按升序排列。bsearch()的原型是：

void *bsearch(const void *key, const void *base, size_t num, size_t width, int (*cmp) (const void *element1, const void *element2));

参数 key 是一个指针，指向要查找的数据项。bsearch()函数返回一个 void 指针，指向第一个与键值匹配的数组元素。

二分法的工作原理是：

将键值与数组正中间的元素进行比较。如果匹配就结束查找；如果小于数组正中间元素，则到数组前半部分继续查找；如果大于数组正中间元素，则到数组后半部分继续查找。以此类推，直到最后找到这个元素。

因此，运用公式。公式：$\log_2{}^{3000000} \approx 21$

面试试题 4

如果使用的下标超出了数组声明的界限，将发生什么情况？

解答

程序将能够通过编译，甚至能够运行，但是结果是不可预测的，这会造成溢出问题。

分析

C 语言运行时常见的溢出错误主要分为堆溢出、栈溢出、BSS（非初始化数据段）溢出及格式化字符串溢出等。如果发生内存溢出，例如本题所述，数组超出了分配的内存，则超出部分将覆盖另外的内存空间，从而引发错误。

面试试题 5

输入一句英文，编写程序使之倒序输出。例如输入：I have got a fever.则输出应为：.fever a got have I

解答

本题是当年笔者去微软公司应聘外包测试工程师时碰到的真题。整张试卷都是考 C 语言，前面是选择题，最后只有这一道编程题。

现在我输入：

I am a MSc student from University of Glasgow and I major in Communication Engineering.

希望输出是：

.Engineering Communication in major I and Glasgow of University from student MSc a am I

解法 1：

Offer!

说明

编程思想：

（1）用 getchar 输入一句话，将其存成一维数组（包括空格），遇回车或换行符时停止。

（2）建立二维数组，每个字符串存成一个数组，这样即可将刚才的一维数组转存为一个二维数组。

（3）按一维逆序、二维顺序输出这个数组。

程序如下：

```
#include <stdio.h>
#include <stdlib.h>

int i,j,cnt,cnt1;
char ch;
char input[256],output[256][256];

int main(void)
{
    i=0;
    cnt=0;
    cnt1=0;

//从键盘输出一句话，存为一维数组
  while(1)
  {
      ch=getchar();
      if((ch==0x0a)||(ch==0x0d))     //遇回车或换行符终止输入
          break;
      input[cnt]=ch;
      cnt++;
  }

//将一维数组转存为二维数组
  j=0;
  for(i=0;i<cnt;i++)
  {
      output[cnt1][j]=input[i];
      if(input[i]==0x20)          //遇空格的处理，一维数字加 1，二维归零
      {
          output[cnt1][j]='\0';  //变一串单字符为字符串，方便今后输出
          cnt1++;
          j=-1;
      }
      j++;
  }
```

Offer

```
// 输出，按照一维逆序二维顺序的原则
  for(i=cnt1;i>=0;i--)
        printf("%s   ",output[i]);

    return 0;
}
```

程序运行结果如图 4-1 所示。

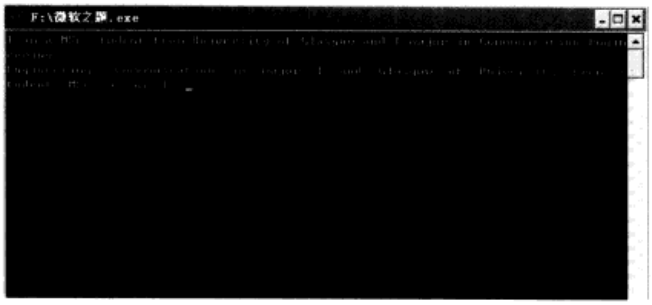


图 4-1　程序运行结果

解法 2：

说明

编程思想：

用指针操作字符串。建立一维数组 input[50]，从屏幕上读取字符串，存入此数组，然后建立二维数组 output[1024][50]，再利用指针操作函数 strcpy(output[i],input)语句将一维数组存入二维数组的第二维（第一维为序号）。当用户输入 3 个星号时，程序停止输入，并倒序打印出每个二维数组的元素——字符串，即倒序打印出每个英文词。

引申

解法 1 是运用单个字符来解，未运用指针；解法 2 抛弃了单个字符的思想，运用指针来操作字符串，程序简洁得多且思维水平较解法 1 高出一个层次。

程序如下：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(void)
{
    char input[50];
    char output[1024][50];
    int   i=0;
```

```
        while(1)
        {
            scanf("%s",input);

    //若输入3个星号，则停止用户输入，开始输出
            if(strcmp("***",input)==0) break;

               else
            {
               strcpy(output[i],input);//指针操作两个字符串首地址，复制字符串
               i++;
            }
        }

    //倒序输出
        for(;i>=0;i--)
            printf("%s ",output[i]);

        printf("\n");
        system("PAUSE");//在 DEV C++编译器环境下，该语句执行系统暂停
        return 0;
}
```

这次我输入一句话是：last time I have forgotten some techniques of C language while at this time I have practised much。

程序运行结果如图 4-2 所示。



图 4-2 程序运行结果

解法 3：

说明

编程思想：

动态分配内存的指针数组解法。当输入3个星号时停止用户输入，开始输出。

引申

解法 3 较前两种解法思维层次更深一步。实际上，对本题而言，这道题不需要如此深的解法，解法 2 是比较合适的一种。因此本解法只是从思维上进行更深一层的探讨。

程序如下：

```
#include <stdio.h>
```

Offer

```
#include <string.h>
#include <stdlib.h>

int main(void)
{
        char *input;
        char *output[1024];
        int   i=0;

//为指针数组动态分配内存，检测返回值
        input=(char *)malloc(50*sizeof(char));
        if(input==NULL) exit(1);

//开始循环并要求用户从屏幕上输入字符串
        while(1)
        {
            scanf("%s",input);

//为指针数组动态分配分存，检测返回值
            output[i]=(char *)malloc(50*sizeof(char));
            if(output[i]==NULL) exit(1);

            if(strcmp("***",input)==0) break; //若输入 3 个星号，则停止用户输入，开始输出
            else
            {
                strcpy(output[i],input);//指针操作两个字符串首地址，复制字符串
                i++;
            }
        }

//倒序输出
        for(i=i-1;i>=0;i--)
            printf("%s ",output[i]);
        printf("\n"); //打印一个空行

//释放内存
        free(input);
        for(;i>=0;i--)
            free(output[i]);

        system("PAUSE");//系统暂停，因为编译器是 DEV C++，这是其内部要求
        return 0;
}
```

这一次我输入一句话：

at this time I have used function-malloc() to solve the problem。



程序运行结果如图 4-3 所示。



图 4-3 程序运行结果

说明

可能有细心的读者已经发现，上述 3 个程序中都存在着同一个 Bug，即逆序输出时，句号并没有出现在最开始，而是位于第 1 个词后面，如第 3 个程序中的“problem.”。这是由于程序中没有对句号进行处理，因此计算机认为“problem”和句号是在同一个字符串之中。

4.4 Java 语言知识考查

Java 是一种面向对象语言，是目前非常流行的网络编程语言之一，它是由美国 Sun Microsystems 公司于 1995 年研发出来的，其前身是 Oak 语言。

在 Java 之前，面向过程语言一直占统治地位，典型的就是 C 语言。C++语言尽管被认为是面向对象语言，实则是介于面向对象与面向过程之间，其保留了面向过程语言的一些特性。

Java 语言具有以下一系列特点。

- 面试对象技术：提高了程序的重用性。
- 健壮性：强类型语言，不允许指针或指针数组访问内存。
- 多线程：这是 Java 的并发机制，同步共享及处理不同事件，使实时交互容易实现。
- 安全性：严格的访问权限设置。
- 分布式计算：Java 类库可处理 TCP/IP 协议，用户可通过 URL 访问网络上的对象。
- 跨平台：Java 解释器采用生成与体系结构无关的字节代码指令技术，不论何种操作系统，安装 Java 运行系统后，Java 程序便可运行。
- 可移植性：运行系统的虚拟机用 C 语言实现，Java 类库可移值，Java 具有跨平台特性，故 Java 本身拥有可移值性。
- 动态性：类库可自由增加方法和实例。Java 以接口支持多重继承，可插入构件和数据库。
- 高性能：Java 字节码可直接转换成对应于特定 CPU 的机器码。
- Applet：Java 的特殊应用程序，嵌在 HTML 中，实现复杂计算或多媒体用户界面。
- 解释执行：用字节码解释执行，字节本身带有很多编译时产生的信息，连接过程简单。

Offer

Java 的实现机制。

- Java 虚拟机：Java 程序首先由编译器转换为标准字节代码，然后由虚拟机解释执行。虚拟机可由软件或硬件实现，其特点是多线程、动态连接及异常处理。
- 垃圾回收：Java 有一个系统级线程，用于跟踪内存使用进行垃圾自动回收，解放了程序员。
- 代码安全检测：在编译器有语法检查，在解释层有字节码校验器，在平台层通过配置策略设定访问资源域。

Java 的体系结构。

最下层是移植接口，由配置器和 Java OS 组成。然后是虚拟机，其上层是 Java 基本类型和基本 API。最上层是 Java 应用程序和 Applet 小程序。

Java 的常用编译器。

主要是 Eclipse 和 JBuilder，另外还有 NetBeans、UltraEdit、Editplus、Jcreator 等。

4.4.1 常见考点汇总

- 多态、重载、覆盖的概念。
- 代码安全检测。
- 跨平台及安全性。
- 虚拟机原理。
- 垃圾回收机制。
- 多线程与同步、交互、锁。
- 线程与进程关系。
- 线程状态与生命周期如 sleep()和 wait()的区别。
- 对象的串行化。
- 静态方法和静态变量。
- 抽象类及方法。
- 内部类和外部类的继承。
- 接口及实现。
- 公共、私有、匿名、静态间的关系、声明和权限控制。
- Final、finally、finalize 的区别。
- 常用公共方法，如 Math.round()。
- 父类、子类的实例化。
- 错误与异常处理。
- 值传递和引用传递。
- 运算符和表达式优先级。
- 条件、循环、多分支语句。



- 递归。
- Java 源程序结构。
- 数组。
- 文件和流。
- AWT、Swing 基础。
- Java.awt、Java.lang、Java.util、Java.io 包中的常用类。
- Applet 的使用。

4.4.2 经典考题及解析

面试试题 1

Java 语言中的多态、重载、继承、覆盖分别是什么含义？

解答

- 本题考查 Java 作为一种面向对象语言的一些基本特性。本题基本上在每次有关 Java 的面试中都会问到。如果本题答错，则基本上面试官就会认为你的 Java 水平接近于 0。
- 多态：Polymorphism，方法的覆盖（重写）、重载与动态连接/调用构成多态性。

说明

多态的表现有如下几种。

（1）使用父类类型的引用指向子类的对象，该引用只能调用父类中定义的方法和变量。

（2）如果子类中重写了父类中的一个方法，那么在调用该方法时，将会调用子类中的这个方法（动态连接/调用）。

（3）变量不能被重写（覆盖），“重写”的概念只针对方法，如果在子类中“重写”了父类中的变量，那么在编译时将会报错。

- 重载：Overloading，在同一类中，多个同名方法同时存在，具有不同的参数个数/类型（和返回类型是否相同无关）。重载是一个类中多态性的一种表现。调用方法时通过传递给它们的不同参数个数和参数类型来决定具体使用哪个方法。
- 继承：Inheritance，继承是面向对象编程技术的一块基石，它允许创建分等级层次的类。通过继承可以创建一个通用类，它定义了一系列相关项目的一般特性。该类可以被更具体的类继承，每个具体的类都可加入自身特性，是父类的一个专门用途的版本。

说明

被继承的类称为超类（superclass），继承超类的类称为子类（subclass）。

这样也提高了代码可复用性。在 Java 中没有多继承，一个类只能有一个父类。

Offer

- 覆盖：Overriding，有的书上也称为“重写”。如果在子类中定义某方法与其父类有相同的名称和参数，则说明该方法被覆盖。当子类的对象使用该方法时，将调用子类中的定义，而非调用父类中的方法。覆盖是多态性的一种表现。

面试试题 2

下列有关 Java 语言的说法正确的是。

A．Abstract 关键字修饰的方法，不能再被子类重写。

B．一个类中包含了 abstract 方法，该类可以不声明为 abstract 类。

C．用 final 关键字修饰的方法，不能再被子类重写。

D．抽象类中一定要包含 abstract 方法。

解答

答案：C。

- 本题考查类中的关键字。
- 用 final 修饰的方法不能再被子类重写。final、finally、finalize 的区别如下：final 用于声明属性、方法和类，分别表示属性不可变、方法不可覆盖、类不可继承；finally 是异常处理语句结构的一部分，表示总是执行；finalize 是 Object 类的一个方法，在垃圾收集器执行时会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。
- Abstract 关键字修饰的方法称为抽象方法，该方法必须被重写。抽象类中不一定要包含 abstract 方法，但是一旦某个类中包含了 abstract 方法，则该类就必须声明为 abstract 类。

面试试题 3

在 Java 语言中，下列说法正确的是。

A．多个方法具有相同的名称，但这些方法的参数类型不同，这属于方法重载。

B．多个方法具有相同的名称，但这些方法的参数个数不同，这属于方法重载。

C．多个方法具有相同的名称，但这些方法的返回类型不同，这属于方法重载。

D．方法重载是指多个方法具有不同的名称，但这些方法的参数类型和个数完全相同。

解答

答案：A、B。

- 本题考查方法重载。方法重载是指多个方法可以具有相同的名称，但这些方法的参数必须不能完全相同，要么参数的类型不同，要么参数的个数不同。
- 如果在两个方法的声明中参数的类型和个数都相同，只是方法的返回类型不同，编译时会产生错误，这表明返回类型不能用来区分重载的方法。



- 注意分区重写和重载：重写是继承后重新实现父类的方法，参数类型和个数都相同；而重载是参数类型或个数不同。

面试试题4

```
import java.io.*;
public class HelloWorld
{
System.out.println("Hello World!");
}
```

以上 Java 程序段的运行结果是：

A．不能通过编译

B．没有输出

C．Hello World!

D．hello World!

解答

答案：A。

- 第一行程序“import java.io.*;”，按需类型导入，导入 java.io 下面的类。所谓按需类型导入，即并非导入整个包，而是导入当前类需要使用的类。

说明

导入有以下两种类型。

（1）单类型导入(single-type-import)，例如 import java.io.File。

（2）按需类型导入(type-import-on-demand)，例如 import java.io.*。

导入是为了简化输入和方便阅读，同时也可以帮助程序员“偷懒”，程序员在使用类名时，不用写类的全路径而只需直接写类名即可。import 不像 C 语言中的#include，会将文件内容都放入进来。不用 import 也无妨，在使用类时写明全路径也可。

说明

java.io 中的类。

（1）按流的方向分为：输入流和输出流。

（2）按流的数据单位不同分为：字节流和字符流。

（3）按流的功能不同分为：节点流和处理流。

- System.out.println()是常用的静态方法，其功能是输出文本。

说明

System 是属于 java.lang 的类，之所以没有 import java.lang，是因为 java.lang 是默认被导入的。



静态方法可以用类名直接调用，即：类名.静态方法名，而非静态方法则必须要获得类的实例后才能以“实例名.方法名”这样的方式调用。Out 是类 System 的变量，是个类，属于 PrintStream 类，也可以说它是一个流类型的静态变量。由于静态变量也可直接用“类名.静态变量名”方式使用，所以其也可以写为 System.out。Println()方法是 PrintStream 类的一个方法，其原型是 void println(Object x)，其中 x 是要打印的对象。

- 由于 Java 区分大小写，所以 public 不能写为 Public。
- Application 程序中必须有 main()方法。
- 和 C 语言不一样，在 Java 语言中，语句不能直接定义在类中，而必须定义在类的方法中。
- 该程序不能编译通过，在 Eclipse 中的运行结果如图 4-4 所示。

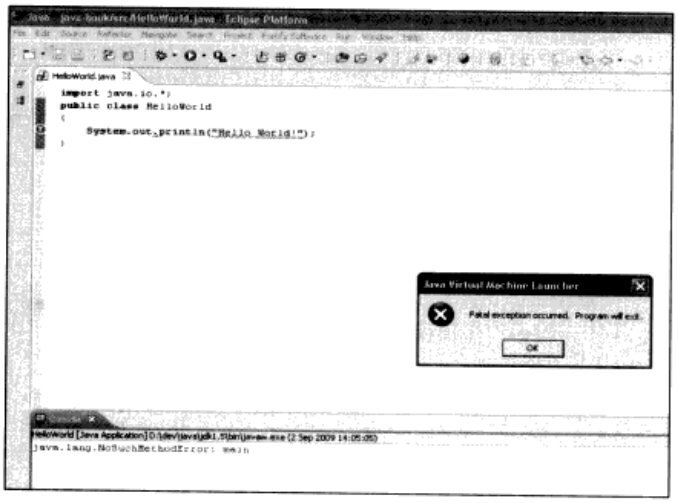


图 4-4 面试试题 4 程序的运行结果

- 正确的程序应当是：

```
import java.io.*;
public class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello World!");
    }
}
```

运行结果如图 4-5 所示。

引申

使用 Eclipse 调试 Java 程序的步骤。

（1）安装软件。安装前需安装 JDK。

（2）工作空间设置：单击 eclipse.exe 启动，设置 Eclipse 的 workspace（工作空间），这是

用户未来建立的Java项目的默认保存路径。

（3）建立Java项目：New→Project→Java Project

（4）加入Java文件：New→Class，每个文件中可以有多个类，但只能有一个public类，文件名同public类的名字。例如public类名为HelloWorld，文件名处就必须写为HelloWorld，保存后，系统自动添加扩展名为.Java，所以最后生成的文件名为HelloWorld.java。

（5）写代码：在文件中输入源程序。

（6）运行：在任务栏中单击图标。

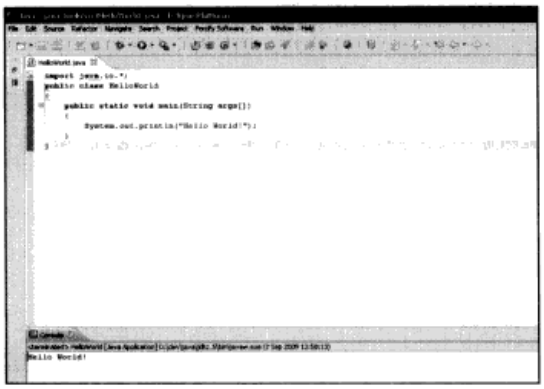


图4-5 面试试题4程序修正后的运行结果

面试试题5

```
class  Media
{
     int i;
     Media()
     {
          add(1);
}
     void print()
     {
          System.out.println(i);
     }
     void add(int v)
     {
          i+=v;
}
}

class Phone extends Media
{
     Phone()
     {
```

Offer!

```
            add(2);
    }
    void add(int v)
    {
        i+=v*2;
    }
}

public class Test
{
    public static void main(String args[])
    {
        apply(new Phone());
    }
    static void apply(Media book)
    {
        book.add(8);
        book.print();
    }
}
```

上述 Java 程序段的输出结果是：

A. 9

B. 13

C. 21

D. 22

E. 24

解答

答案：D。

- 运行结果如图 4-6 所示。

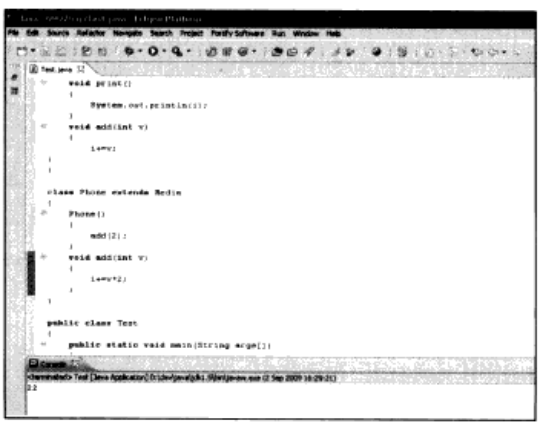


图 4-6 面试试题 5 程序及运行结果

Offer

- 本题考查构造函数用法。因此，Java 在类里提供了一个特殊的成员函数，称为构造函数（Constructor），主要用于在创建对象时初始化对象。

说明

构造方法是一种特殊的方法，其特点如下。

（1）构造方法的方法名必须与类名相同。

（2）构造方法没有返回类型，也不能定义为 void，在方法名前面不声明方法类型。

（3）构造方法的主要作用是完成对象的初始化工作，它能够把定义对象时的参数传给对象的域。

（4）构造方法不能由编程人员调用，而需系统调用。

（5）一个类可以定义多个构造方法，如果在定义类时没有定义构造方法，则编译系统会自动插入一个无参数的默认构造器，这个构造器不执行任何代码。

（6）构造方法可以重载，以参数的个数、类型或排列顺序区分。

- 题目程序段创建的是 Phone 类的实例，在运行时，所有调用 add()方法的过程将始终和 Phone 类的 add()方法动态绑定。
- Media book= new Phone()，父类对象可以引用子类实例，反之不可以。这个实例化过程是先调用父类中的构造方法，然后再执行子类中的构造方法。

说明

它表示用户定义了一个 Media 类型的引用，并指向新建的 Phone 类型的对象。由于 Phone 是继承自它的父类 Media，所以 Media 类型的引用是可以指向 Phone 类型的对象的。那么这样做有什么意义呢？因为子类是对父类的一个改进和扩充，所以一般子类在功能上较父类更强大，属性较父类更独特，定义一个父类类型的引用指向一个子类的对象既可以使用子类强大的功能，又可以抽取父类的共性。所以，父类类型的引用可以调用父类中定义的所有属性和方法，但对于子类中定义而父类中没有的方法，它是“无可奈何”的。同时，父类中的一个方法只有在父类中定义而在子类中没有重写的情况下，才可以被父类类型的引用调用。对于父类中定义的方法，如果子类中重写了该方法，那么父类类型的引用将会调用子类中的这个方法。

- i 初始值是 0。创建实例 new Phone()，先调用父类的默认构造函数 Media()，并在父类的默认构造函数中执行 add(1)，i=0+1×2=2，再调用子类的默认构造函数 Phone()，子类的默认构造函数中执行 add(2)，i=2+2×2=6，执行 add(8)，i=6+8×2=22。

说明

应聘者若能向面试官描述清楚本题的各个步骤，则说明他对 Java 的构造函数概念非常清晰，理论上属于中级以上水平。

Offer

4.5 UNIX Shell 知识考查

UNIX Shell 也称为命令行界面，它是 UNIX 操作系统下传统的用户和计算机的交互界面。Shell 用于解释在交互方式下用户输入的命令行命令，同时它还可以定制用户环境，是解释性的编程语言。

普通意义上的 Shell 就是指可以接受用户输入命令的程序。它之所以被称做 Shell 是因为它隐藏了操作系统底层的细节。同样，对于 UNIX 下的图形用户界面 GNOME 和 KDE，有时也被称为虚拟 Shell 或图形 Shell。

UNIX 操作系统下的 Shell 既是用户交互的界面，也是控制系统的脚本语言。当然在这点也有别于 Windows 下的命令行，虽然命令行也提供了很简单的控制语句。

在 Windows 操作系统下，可能有些用户从来都不会直接使用 Shell，然而在 UNIX 操作系统下，Shell 是控制系统启动、X Window 启动、很多其他实用工具的脚本解释程序，用户会直接使用它。

基于 Algol 语言，Bourne shell 于 1979 年由 Stephen Bourne 研发，其主要用于系统管理自动化，但缺少很多交互功能。

基于 C 语言，C shell 由美国加州大学伯克利分校在 20 世纪 70 年代末开发，注重了交互功能，但是脚本执行速度不如 Bourne shell。

AT&T 的 David Korn 于 1986 年发布了 Korn shell，是 Bourne shell 的扩展集，并加入 C shell 中一些功能。

Brian Fox 于 1988 年推出了 bash（Bourne Again shell）。这是 Linux 操作系统上默认的 shell。Bourne shell 脚本可在 bash 下运行，另外，bash 还集成了 C shell 和 Korn shell 很多有用的功能。

TC shell 是 C shell 的扩展版本。

4.5.1 常见考点汇总

- Shell 的定义与功能。
- Shell 内置命令。
- Shell 编程的位置参量。
- Shell 的命令行参数。
- 标准 I/O 和重定向。
- 正则表达式与模式匹配。
- Grep 命令。
- 流编辑器 Sed。
- Awk 使用。



4.5.2 经典考题及解析

面试试题1

使用 UNIX Shell 中的 grep 命令在 Linux 系统中实现下列查找。
（1）对文本文件 datafile，显示所有包括 tom 或 Tom 的行，设置 grep 为大小写不敏感。
（2）显示所有不包括 lily 的行。
（3）显示所有以 M 开头、以 t 结尾的行。
（4）显示所有这样的行，它包括一个词，这个词以一个大写字母开头，后跟 3 个小写字母。

解答

（1）grep –i '[Tt]om' datafile
（2）grep –v 'lily' datafile
（3）grep '^Mt$' datafile
（4）grep '[A-Z][a-z][a-z][a-z]' datafile

面试试题2

请分别解释 bash shell 中标准 I/O 和重定向, exec 命令和重定向。

解答

shell 启动时继承了 3 个文件：stdin、stdout 和 stderr。标准输入通常来自键盘。标准输出和标准错误输出通常被发往屏幕。有时需要从文件读取输入，或者将输出结果和报错信息写入文件。这些都可以通过 I/O 重定向来实现。

exec 命令能够在不启动新进程的前提下，将当前正在运行的程序替换为一个新的程序。使用 exec 命令，不需创建子 Shell 就能改变标准输入和输出。使用 exec 命令打开文件后，read 命令每次都会将文件指针移到文件的下一行，直到文件末尾。如果要再次从头开始读文件，就必须先关闭文件再打开。

标准 I/O 重定向表和 Exec 命令表如表 4-1 和表 4-2 所示。

表 4-1 标准 I/O 重定向表

重定向操作符	功 能
< 文件名	重定向输入
> 文件名	重定向输出
>>文件名	追加输出
2>文件名	重定向标准错误输出
2>>文件名	重定向和追加标准错误输出

Offer

续表

重定向操作符	功　能
&>文件名	重定向标准输出和标准错误输出
>&文件名	重定向标准输出和标准错误输出(首选方式)
2>&1	将标准错误输出重定向到输出的去处
1>&2	将输出重定向到标准错误输出的去处
>\|	重定向输出时忽略 noclobber
<>文件名	如果是一个设备文件(/dev)，则使用文件作为标准输入和标准输出

表 4-2　　exec 命令表

exec 命令	功　能
exec ls	ls 将顶替 shell 运行，当其运行结束后，它启动时所在的 shell 不会返回运行状态
exec <filea	打开文件 filea，用于读标准输入
exec >filex	打开文件 filex，用于写标准输出
exec 3<datfile	打开文件 datfile，将其作为文件描述符 3，用于读取输入
sort <&3	将文件 datfile 排序
exec 4>newfile	打开文件 newfile，将其作为文件描述符 4，用于写输出
ls >&4	将 ls 的输出结果重定向到 newfile
exec 5<&4	将文件描述符 5 变成文件描述符 4 的一个副本
exec 3<&-	关闭文件描述符 3

面试试题 3

请谈谈对 shell 脚本的认识并说出如何创建 shell 脚本?

解答

shell 脚本简介。

当命令不在命令行上执行而是通过一个文件执行时，该文件就称为 Shell 脚本，脚本以非交互的方式运行。当 bash shell 以非交互方式运行时，它先查找环境变量 BASH_ENV（或 ENV），该变量指定了一个环境文件（通常是.bashrc），然后从该文件开始执行。当读取了 BASH_ENV 文件后，shell 就开始执行脚本中的命令。

创建 shell 脚本步骤。

shell 脚本通常在编辑器中编写，由命令及其注释组成，注释是跟在#后面的内容，用来对脚本进行注解。

（1）第一行：称为 shbang 行，写为#!/bin/bash。

（2）注释：注释是跟在#号后的行。

（3）可执行语句与 bash shell 结构：一个 bash shell 程序由一组 UNIX/Linux 命令、bash shell

Offer!

命令、程序结构控制语句和注释组成。

（4）使脚本可执行：如果要运行脚本，就必须赋予它可执行的权限。此时可以使用 chmod 命令来打开脚本的执行权限。

面试试题4

请解释 shell 中元字符的含义，如‘/’，‘*’，‘？’，‘(cmds)’等。

解答

元字符是一种可以用来代表自身以外的内容的特殊字符。shell 的元字符也被称做“通配符”。在 shell 命令或脚本中经常会用到元字符，所以对元字符的掌握可以说是应用 shell 的一个必备的前提。Shell 中常用的元字符如表 4-3 所示。

表 4-3 shell 中常用的元字符

元 字 符	含 义
\	按字面含义解释它后面那个字符
&	在后台处理的进程
;	分隔命令
$	替换变量
?	匹配单个字符
[abc]	匹配这组字符中的一个。例如，a、b 或 c
[!abc]	匹配这组字符以外的某个字符。例如，除 a、b 或 c 以外的字符
*	匹配零个或多个字符
(cmds)	在子 shell 中执行命令
{cmds}	在当前 shell 中执行命令

面试试题5

谈谈对 Shell 流编辑器 Sed 的认识并对 Sed 常用命令简单进行介绍。

解答

Sed 简介。

Sed 是流线型、非交互式编辑器，它允许用户执行与 vi 和 ex 编辑器中相同的编辑任务。Sed 程序不是与编辑器交互式工作的，而是让用户在命令行中输入编辑的命令，为文件命名，然后在屏幕上查看命令输出结果。Sed 编辑器按一次处理一行的方式来处理文件，并把输出送到屏幕上。

Sed 常用命令。

- a\ 在当前行上添加一个文本行或多个文本行。
- c\ 用新文本改变（取代）当前行中的文本。

Offer

- d 删除行。
- i\ 在当前行之前插入文本。
- h 把模式空间内容复制到一个固定缓存。
- H 把模式空间内容添加到一个固定缓存。
- g 得到固定缓存中的内容并复制到模式缓存，重写其内容。
- G 得到固定缓存的内容并复制到模式缓存，添加到里面。
- I 列出不打印的字符。
- p 打印行。
- n 读下一输入行，并开始用下一个命令处理换行符，而不是用第一个命令。
- q 结束或退出 Sed。
- r 从一个文件读入行。
- ! 把命令应用到除了选出的行以外的其他所有行。
- s 把一个字符串替换成另一个。
- g 在一行上进行全局替换。
- p 打印行。
- w 把行写到一个文件中。
- x 用模式空间的内容交换固定缓存的内容。
- y 把一个字符转换成另一个（不能和正则表达式元字符一起使用）。

强调

Sed 流编辑器是 Shell 编程中常用的一个编辑器，其功能十分强大，用户应掌握最为常用的 Sed 命令，如。

- **a\，在当前行上添加一个文本行或多个文本行。**
- **s，把一个字串替换成另一个。**

当然，如果能将常用命令全部掌握最好。

4.6 网络知识考查

计算机网络，是指将地理位置不同的具有独立功能的多台计算机及其外部设备，通过通信线路连接起来，在网络操作系统、网络管理软件及网络通信协议的管理和协调下，实现资源共享和信息传递的计算机系统。关于计算机网络的最简单定义是：一些相互连接的、以共享资源为目的的计算机的集合。

按计算机连网的区域大小，我们可以把网络分为局域网（LAN，Local Area Network）和广域网（WAN，Wide Area Network）。局域网（LAN）是指在一个较小地理范围内的各种计算机

Offer!

网络设备互联在一起的通信网络，其可以包含一个或多个子网，通常局限在几千米的范围之内。例如在一个房间、一座大楼或是在一个校园内的网络就称为局域网。广域网（WAN）连接地理范围较大，常常是一个国家或是一个洲，其目的是为了让分布较远的各局域网互联。例如我们平常讲的 Internet 就是最大、最典型的广域网。

计算机网络的功能主要表现在硬件资源共享、软件资源共享和用户间信息交换三个方面。硬件资源共享，软件资源共享和用户间信息交换。

4.6.1 常见考点汇总

- 因特网定义。
- 因特网发展的 3 个阶段。
- 因特网的标准化。
- 因特网的组成。
- 因特网的核心部分。
- 计算机网络的分类。
- 计算机网络的性能。
- 计算机网络的体系结构。
- 计算机网络体系结构的形成。
- 协议与划分层次。
- 具有五层协议的体系结构。
- 实体、协议、服务和服务访问点。
- TCP/IP 体系结构。
- 物理层。
- 数据链路层。
- 网络层。
- 运输层。
- 应用层。
- 网络安全。
- 无线网络。
- 下一代因特网。

4.6.2 经典考题及解析

面试试题 1

请解释一下计算机网络中 OSI 七层参考模型。



解答

在计算机网络产生之初，每个计算机厂商都有一套自己的网络体系结构的概念，它们之间互不相容。为此，国际标准化组织（ISO）于1979年建立了一个分委员会来专门研究一种用于开放系统互联的体系结构（Open Systems Interconnection）简称OSI，“开放”这个词表示：只要遵循OSI标准，某个系统就可以和位于世界上任何地方也同时遵循OSI标准的其他任何系统进行连接。

OSI参考模型分为7层，分别是物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。

各层的主要功能及其相应的数据单位如下。

- 物理层：通常把物理媒体当作第0层，物理层的任务就是为它的上一层提供一个物理连接以及其机械、电气、功能和过程特性。在这一层，数据还没有被组织，仅作为原始的位流或电气电压处理，单位是比特。
- 数据链路层：该层负责在两个相邻节点间的线路上，无差错地传送以帧为单位的数据。每一帧包括一定数量的数据和一些必要的控制信息。
- 网络层：该层的任务就是选择合适的网间路由和交换节点，以确保数据能够及时传送。网络层将数据链路层提供的帧组成数据包，包中封装有网络层包头，其中含有逻辑地址信息。
- 传输层：该层的任务是根据通信子网的特性最佳地利用网络资源，并以可靠和经济的方式，为两个端系统（也就是源站和目的站）的会话层之间提供建立、维护和取消传输连接的功能，负责可靠地传输数据。
- 会话层：该层也可以称为会晤层或对话层，其不参与具体的传输，提供包括访问验证和会话管理在内的建立和维护应用之间通信的机制。例如服务器验证用户登录便是由会话层完成的。
- 表示层：该层主要解决用户信息的语法表示问题。提供格式化的表示和转换数据服务。数据的压缩和解压缩以及加密和解密等工作都由表示层负责。
- 应用层：该层确定进程之间通信的性质，以满足用户需要以及提供网络与用户应用软件之间的接口服务。

说明

在熟悉OSI七层模型的同时，我们也应该掌握TCP/IP模型。TCP/IP模型主要包括以下几个层次：互联网层、传输层、应用层、主机至环联网层。各层含义在此就不做赘述。OSI参考模型和TCP/IP模型有很多共同点：例如两者都是以协议栈的概念为基础，并且协议栈中的协议彼此相互独立，另外两个模型中各个层的功能也大体相似。当然，两个模型也有很多不同点：例如OSI模型中的协议比TCP/IP模型中的协议有更好的隐蔽性，当技术发生变化时OSI模型中的协议相对更加容易被替换成新的协议。

Offer

面试试题2

请解释什么是TCP/IP并分别解释其中主要协议IP、TCP、UDP、ICMP的各自功能以及工作原理。

解答

TCP/IP（Transmission Control Protocol/Internet Protocol），中文译名为传输控制协议/因特网互联协议，又称为网络通信协议，该协议是Internet最基本的协议——Internet国际互联网络的基础，简单地说，就是由网络层的IP协议和传输层的TCP协议组成的。

TCP/IP是一个两层的程序。高层为传输控制协议，它负责聚集信息或把文件拆分成更小的包。这些包通过网络传送到接收端的TCP层，接收端的TCP层把包还原为原始文件。低层是网际协议，它负责处理每个包的地址部分，使这些包正确地到达目的地。网络上的网关计算机根据信息的地址来进行路由选择。

1．IP：网际协议

IP是TCP/IP的“心脏”，也是网络层中最重要的协议。

- IP层接收由更低层发来的数据包，并把该数据包发送到更高层TCP或UDP层。IP数据包是不可靠的，因为IP并没有做任何事情来确认数据包是按顺序发送的或者没有被破坏。
- 高层的TCP和UDP服务在接收数据包时，通常假设包中的源地址是有效的。

2．TCP：传输控制协议

如果IP数据包中有已经封好的TCP数据包，那么IP将把它们向‘上’传送到TCP层。TCP将包排序并进行错误检查，同时实现虚电路间的连接。由于TCP数据包中包括序号和确认码，所以未按照顺序收到的包可以被排序，而损坏的包则可以被重传。

TCP将它的信息送到更高层的应用程序，例如Telnet的服务程序和客户程序。应用程序轮流将信息送回TCP层，TCP层便将它们向下传送到IP层、设备驱动程序和物理介质，最后到接收方。

3．UDP：用户数据报协议

UDP与TCP位于同一层，但它不负责数据包的顺序、错误或重发，因此UDP不被应用于那些使用虚电路的面向连接的服务，而主要用于那些面向查询——应答的服务，例如NFS。使用UDP的服务包括NTP（网络时间协议）和DNS（DNS也使用TCP）。

4．ICMP：互联网控制消息协议

ICMP与IP位于同一层，它被用来传送IP的控制信息以及提供有关通向目的地址的路径信息。ICMP的“Redirect”信息通知主机通向其他系统的更准确的路径，而“Unreachable”信息则指出路径有问题。另外，如果路径不可用，则ICMP可以使TCP连接“体面地”终止。PING是最常用的基于ICMP的服务。



面试试题3

请描述路由器的定义及作用。

解答

路由器是用来实现路由选择功能的一种媒介系统设备。所谓路由就是指通过相互连接的网络把信息从源地点移动到目标地点的活动。一般来说，在路由过程中，信息至少会经过一个或多个中间节点。

路由器的作用主要是连通不同的网络和选择信息传送的线路。从过滤网络流量的角度来看，路由器的作用与交换机和网桥非常相似。选择最佳路径的策略即路由算法是路由器的关键所在。为了完成这项工作，在路由器中保存着各种传输路径的相关数据即路径表，供路由选择时使用。

主要路径表有以下两种。

- 静态路径表：是指系统管理员事先设置好的固定的路径表。它一般在系统安装时就根据网络的配置情况预先设定，不会随着未来网络结构的改变而改变。
- 动态路径表：是路由器根据网络系统的运行情况而自动调整的路径表。路由器根据路由选择协议提供的功能，自动学习和记忆网络运行情况，在需要时自动计算数据传输的最佳路径。

面试试题4

请列出网络拓扑结构的类型并分别进行解释。

解答

网络拓扑结构是指用传输媒体互联各种设备的物理布局，也就是说，用什么方式把网络中的计算机等设备连接起来。

目前大多数网络使用的拓扑结构主要有以下 3 种。

（1）星型拓扑结构：星型拓扑结构是用一个节点作为中心节点，其他节点直接与中心节点相连构成的网络。中心节点既可以是文件服务器，也可以是连接设备。常见的中心节点为集线器。星型拓扑结构的网络属于集中控制型网络，整个网络由中心节点执行集中式通行控制管理，各节点间的通信都要通过中心节点。

（2）环型拓扑结构：环型拓扑结构是使用公共电缆组成一个封闭的环，各节点直接连到环上，信息沿环按着一定的方向从一个节点传送到另一个节点。环接口一般由发送器、接收器、控制器、线控制器和线接收器组成。

（3）总线型拓扑结构：总线型拓扑结构的网络只有一条唯一的电缆干线，以菊链的形式连接一个接一个工作站。在实际的安装中，电缆线穿过一个接一个的办公室。所有的节点共享同一介质，某一时刻只有一个节点能够广播消息。

Offer!

面试试题5

请问你是否设计实现过网络系统？如果是，请阐述网络系统设计的原则。

解答

网络系统的设计应遵循以下几条原则。

（1）实用性：应当从实际情况出发，采用成熟的技术和产品来建设该系统。要能将新系统与已有的系统兼容，保持资源的连续性和可用性。系统安全可靠，使用方便，不需要太多的培训即可容易的使用和维护。

（2）先进性：采用当前国际先进成熟的主流技术和业界的相关国际标准。设备选型要是先进和系列化的，系统应是可扩充的。

（3）安全性：采用各种有效的安全措施，保证网络系统和应用系统安全运行。安全包括4个层面：网络安全、操作系统安全、数据库安全、应用系统安全。

（4）可扩充性：采用符合国际和国内工业标准的协议和接口，从而使网络具有良好的开放性，实现与其他网络和信息资源的容易互联互通，并可以在网络的不同层次上增加节点和子网。

（5）可管理性：设计网络时充分考虑网络日后的管理和维护工作，并选用易于操作和维护的网络操作系统，以减轻网络运营时的管理和维护负担。

4.7 质量管理知识考查

很多客户在选择软件企业/组织时，都要求软件企业/组织通过一定级别的CMMI认证。放眼望去，国内大部分软件公司都在努力地为CMMI认证在奋斗。那么CMMI到底是什么？

为了统一质量管理标准，扫除国际贸易的障碍，国际化标准组织（ISO）于1979年成立了质量管理和质量保障技术委员会（TC176）。随后TC176发布了ISO9000等系列标准，质量认证体系逐步在全世界范围内得到采用。

质量管理体系可以帮助组织提高客户满意度，促使组织持续改进产品及流程，并使整个过程可控。

质量管理体系方法是以过程方法为基础，以持续改进为阶梯，建立适合组织的有效运行的一套系统严谨的逻辑步骤和运作程序。过程是质量管理活动研究的基本单元。

IS09000族标准在多年实施中确立了8项质量原则。

- 以客户为中心。
- 领导作用。
- 全员参与。
- 过程方法。

Offer

- 管理的系统方法。
- 持续改进。
- 基于事实的决策方法。
- 与供方互利的关系。

质量方针是由组织的最高管理者正式发布的该组织总的质量宗旨和方向；质量目标是指组织在质量方面所追求的目的，其应是可测的。

软件工程中的质量管理（SQA）通常包括以下几个方面。

- 软件质量保障流程。
- 项目管理流程。
- 软件开发流程。
- 软件测试流程。
- 软件配置管理流程。

美国卡内基梅隆大学于 1991 年推出 CMM（软件能力成熟度模型）1.0 版，后又于 2000 年推出 CMMI（软件能力成熟度模型集成）1.0 版，对软件供应商能力进行评估。

CMMI 有连续式（0～5 级）和阶段式（1～5 级）两种：连续式采用能力等级来度量过程改进的程度；阶段式采用成熟度级别划分表示组织整体的成熟情况。

连续式 CMMI 能力等级如表 4-4 所示。

表 4-4 连续式 CMMI 能力等级

能力级别	连续式表示能力级别
0	不完整级（Incomplete）
1	已执行级（Performed）
2	已管理级（Managed）
3	已定义级（Defined）
4	已定量管理级（Quantitatively Managed）
5	优化级（Optimizing）

阶段式 CMMI 能力等级如表 4-5 所示。

表 4-5 阶段式 CMMI 能力等级

成熟度级别	阶段式表示成熟度级别
1	初始级（Initial）
2	已管理级（Managed）
3	已定义级（Defined）
4	已定量管理级（Quantitatively Managed）
5	优化级（Optimizing）

在质量管理中，软件过程改进占据了重要位置。



说明

CMMI所描述的是软件开发方法“是什么”，而软件过程改进的目标则是具体到“如何做”。

软件过程是实施于软件开发和维护中的阶段、方法、技术、实践及相关产物（计划、文档、模型、代码、测试用例和手册等）的集合。

软件过程促进开发工作的标准化，提高开发效率，促进组件的可重用性和一致性，引进业内先进实践技术，改进软件维护和技术支持。总而言之，软件过程使企业对软件工程的管理更加规范、有序、合理。

流行的软件过程主要包括以下几种。

- RUP，即 Rational Unified Process。
- OPEN Process。
- OOSP，即 Object-Oriented Software Process。
- XP（极限编程），即 eXtreme Programming。
- Catalysis。
- DSDM，Dynamic System Development。

作为软件测试工程师，我们在质量管理体系中的作用至关重要，质量管理经理是测试工程师职业发展的一个重要方向。

本节，我们通过5道典型面试笔试题来帮助大家对质量管理领域的考题有所认识。

4.7.1 常见考点汇总

- 质量管理的基本概念（如CMM、CMMI）。
- 软件测试数据分析(包括各种测试数据图表使用)。
- 测试项目的管理包括工具(如微软 Project)使用。
- 配置管理概念（SCM）及工具(如CVS、VSS)使用。
- 软件度量概念（包括产品度量、过程度量、项目度量）。
- 软件过程改进（如RUP、XP、OOSP）。
- 过程改进方法（如PDCA、IDEAL）。

4.7.2 经典考题及解析

面试试题1

戴明循环是什么？有什么意义？

解答

即PDCA循环，又称为戴明环。它是由美国的William Edwards Deming最早提出来的。

- P（Plan）：计划，包括方针和目标的确定以及活动计划的制定。

Offer

- D（DO）：执行，具体运作，实现计划中的内容。
- C（Check）：检查，总结执行计划的结果，分清哪些正确，哪些错误，明确效果，找出问题。
- A（Action）：行动，对总结进行处理，成功处加以标准化，总结失败。对于未解决问题，提给下一个 PDCA 循环中解决，如图 4-7 所示。

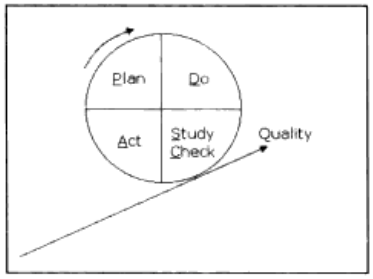


图 4-7 戴明环

PDCA 循环的特点。

- 周而复始：未解决或新的问题，放入下一个循环。
- 大环带小环：组织与其内部子体系的关系，是大环带小环。
- 阶梯式上升。
- 统计工具：PDCA 是推动工作、发现问题和解决问题的有效工具。

PDCA 循环的 8 个步骤。

（1）分析现状，发现问题。

（2）分析质量问题的主要原因。

（3）针对主要原因，采取解决的措施。

（4）制定措施计划。

（5）执行，按措施计划的要求去做。

（6）检查，对比执行结果与目标。

（7）标准化，总结成功经验。

（8）新问题转入下一个 PDCA 循环。

PDCA 循环的 7 种工具：

直方图、控制图、因果图、排列图、相关图、分层法、统计分析表。

面试试题 2

极限编程是什么？

解答

极限编程（eXtreme Programming，简称 XP）是敏捷方法中最著名的一个。它是在 1998 年由 Smalltalk 社群中的大师级人物 Kent Beck 首先倡导的，是软件开发的更简单有效的方法。交流、简单、反馈、勇气是 XP 程序员的 4 个要点。极限编程适合于 2~12 人的小型开发团队。

极限编程的准则包括。

（1）完整团队。XP 项目的所有参与者（开发人员、客户、测试人员等）一起工作在一个开放的场所中，他们是同一个团队的成员。这个场所的墙壁上可随意悬挂着大幅的、显著的图表以及其他一些显示项目进度的东西。

Offer!

（2）规划策略。计划是持续的、循序渐进的。每两周，开发人员就为下两周估算候选特性的成本，而客户则根据成本和商务价值来选择要实现的特性。

（3）客户测试。作为选择每个所期望的特性的一部分，客户可以根据脚本语言来定义出自动验收测试来表明该特性可以工作。

（4）小发行版本：为了高度迭代，与客户展现开发的进展，小版本发布是一个可交流的好办法，客户可以针对性提出反馈。但由于小版本把模块缩得很小，会影响软件的整体思路连贯，所以小版本也需要总体合理的规划。

（5）简单设计。保持设计和当前的系统功能相匹配。它通过了所有的测试，不包含任何重复，表达出了编写者想表达的所有东西，并且包含尽可能少的代码。

（6）成对编程。所有的产品软件都是由两个程序员一同构建的。

（7）测试优先开发。编写单元测试是一个验证行为，也是一个设计行为，同样它更是一种编写文档的行为。该行为避免了相当数量的反馈循环，尤其是功能验证方面的反馈循环。程序员以非常短的循环周期工作，他们先增加一个失败的测试，然后使之通过。

（8）改进设计。随时利用重构方法改进已经“腐化”的代码，保持代码尽可能的干净，具有表达力。

（9）持续集成。团队总是使系统完整地被集成。当一个人检入（Check in）后，将其他所有人的代码集成。

（10）集体代码所有权。任何结对的程序员都可以在任何时候改进任何代码。没有程序员对任何一个特定的模块或技术单独负责，每个人都可以参与任何其他方面的开发。

（11）编码标准。系统中所有的代码看起来就好像是单独一人编写的。

（12）系统比喻。即将整个系统联系在一起的全局视图。它是系统的未来影像，使得所有单独模块的位置和外观变得明显直观。如果模块的外观与整个系统比喻不符，那么就说明该模块存在错误。

（13）可持续的速度。团队只有持久才有获胜的希望。他们以能够长期维持的速度努力工作，保存精力，他们把项目看作是马拉松长跑，而不是全速短跑。

极限编程是一组简单、具体的实践，这些实践结合在形成了一个敏捷开发过程，同时它又是一种优良的、通用的软件开发方法，项目团队可以拿来直接使用，也可以增加一些实践或对其中的一些实践进行修改后再使用。

面试试题3

如何进行软件测试的风险分析？

解答

（1）成立分析小组，包括最终用户、开发人员、测试人员及销售人员的代表。

（2）编制特征列表，即系统各模块的特征和测试策略。



（3）确定失效可能性，即各个模块不能完成任务的失效可能性，给予数值。

（4）确定影响，即模块若失效，对用户影响有多大，给予数值。

（5）确定风险优先级。将失效可能性及影响的数值求和，将各模块按和的从大到小排序。

（6）评审和修改值。从复杂性、环境、资源、经验和历史数据等方面的情况来评审和修改优先级。

（7）确定分割线，将模块分为“待测试”和“不予测试”两类。

（8）考虑缓解风险，确定哪些风险能通过增加资源、改变开发方法等方式规避或缓解。

面试试题 4

为何 CMM 会被 CMMI 取代？

解答

自从 1994 年 SEI 正式发布 CMM 软件以来，相继又开发出了系统工程、软件采购、人力资源管理以及集成产品和过程开发方面的多个能力成熟度模型。虽然这些模型在许多组织都得到了良好的应用，但对于一些大型软件企业来说，可能会出现需要同时采用多种模型来改进自己多方面过程能力的情况。这时他们就会发现存在一些问题，其中主要问题体现在。

- 不能集中其不同过程改进的能力以取得更大成绩。
- 要进行一些重复的培训、评估和改进活动，因而增加了许多成本。
- 遇到不同模型中有一些对相同事物说法不一致或活动不协调，甚至相抵触的情况。

于是，希望整合不同 CMM 模型的需求就产生了。1997 年，美国联邦航空管理局（FAA）开发了 FAA-iCMMSM（联邦航空管理局的集成 CMM），该模型集成了适用于系统工程的 SE-CMM、软件获取的 SA-CMM 和软件的 SW-CMM 3 个模型中的所有原则、概念和实践，因此也被认为是第一个集成化的模型。

CMMI 与 CMM 最大的不同点在于：CMMISM-SE/SW/IPPD/SS 1.1 版本有 4 个集成成分，即：系统工程（SE）和软件工程（SW）是基本的科目，对于有些组织还可以应用集成产品和过程开发方面（IPPD）的内容，如果涉及供应商外包管理还可以相应地应用 SS（Supplier Sourcing）部分。

面试试题 5

RUP 是什么？

解答

统一软件过程（Rational Unified Process，PUP）是由 Rational 软件公司（后被 IBM 公司并购）开发的一套软件过程框架。它贯穿于整个项目生命周期，在管理和工程活动方面为开发团队提供指导，是一个面向对象且基于网络的程序开发方法论。它以用例为驱动，以体系结构为中心，采用迭代和增量的过程，经历 5 种工作流：需求、分析、设计、实现、测试，最终构造

Offer

完善的软件产品。RUP 和类似的产品（例如面向对象的软件过程 OOSP 和 OPEN Process）都是理解性的软件工程工具：把开发中面向过程的方面（例如定义的阶段、技术和实践）和其他开发的组件（例如文档、模型、手册以及代码等）整合在一个统一的框架内。RUP 是一个基于 6 个最佳开发实践的流程定义产品。

6 个最佳开发实践。

（1）迭代式开发。

（2）需求管理。

（3）基于组件的体系架构。

（4）可视化建模。

（5）持续的质量管理。

（6）配置管理。

RUP 是一个流程定义平台，是一个流程框架。

- RUP 是风险驱动的、基于 Use Case 技术的并以架构为中心的、迭代的、可配置的软件开发流程。
- 用户可以针对 RUP 所规定的流程进行客户化定制，定制出适合自己组织的实用的软件流程。

RUP 如何实现 6 个最佳开发实践？请看以下两点。

（1）把软件开发过程看成是多次迭代开发的过程，并且把迭代开发分成 4 个阶段（每个阶段由若干次迭代组成）。

- 开始阶段：定义出项目目标和范围。
- 细化阶段：制定计划、定义项目基线、确定系统的体系架构。
- 构造阶段：主要进行编码、单元测试工作，属于人工最密集的阶段。此时虽然允许有小的需求加入进来，但还是应该尽量避免大的需求变动。
- 交付阶段：将产品提交给用户使用，另外还包括相关的培训等内容。

（2）定义出一次迭代开发需遵循以下 9 个准则。

- 商业建模。
- 需求。
- 分析和设计。
- 实现。
- 测试。
- 部署。
- 配置和变更管理。
- 项目管理。
- 环境。

其中前 6 个准则称为核心工程流，后 3 个准则称为支持工作流。在每次迭代中，流程都要遍历所有的准则。



4.8 Linux 知识考查

Linux 是一类 UNIX 计算机操作系统的统称。严格来讲，Linux 这个词本身只表示 Linux 内核，但在实际上人们已经习惯了用 Linux 来形容整个基于 Linux 内核的操作系统。Linux 操作系统是可以运行在许多不同类型的计算机上的一种操作系统的“内核”，是提供命令行或程序与计算机硬件之间接口的软件的核心部分。Linux 操作系统内核管理以下事情：例如内存、采用什么方法、在何时打开或关闭文件以及哪一个进程或程序可以获得计算机的中央处理单元（CPU）等。

Linux 内核最初只是由芬兰人林纳斯·托瓦兹（Linus Torvalds）在赫尔辛基大学上学时出于个人爱好而编写的，当时他并不满意原有的教学用的操作系统，部分只能在有限硬件上运行。

Linux操作系统继承了UNIX操作系统的超过25年的经验、源代码及技术支持，这使得Linux操作系统虽然是最新操作系统中的一员，却远比当今流行的商业化的操作系统成熟。

Linux 操作系统具有以下几大显著特点。

- 是真正的多用户、多任务、多平台操作系统。
- 提供 Shell 命令解释程序和编程语言。
- 提供强大的管理功能和大量使用程序。
- 具有内核的编程接口和图形用户接口。
- 系统的组成部分的源代码是开放的。

4.8.1 常见考点汇总

- Linux 操作系统的安装。
- Linux 操作系统的配置。
- Linux 操作系统的基本命令。
- 操作文件与搜索命令。
- 了解和使用 Linux shell。
- KDE 桌面环境。
- Linux 系统与因特网的连接。
- 配置因特网电子邮件。
- Linux 编辑器的使用,如 Xemacs、Visual iMproved。
- 排版打印文档。
- Linux 图形工具的使用。
- Linux 操作系统的监控。
- 使用 Cron 程序实现系统管理自动化。

Offer

4.8.2 经典考题及解析

面试试题1

Linux中用来浏览及搜索文件系统的命令有哪些？请列出并分别解释。

解答

在Linux文件系统中移动、查找文件是经常要进行的操作，所以用户对一些常用的浏览及搜索命令要熟练掌握。下面主要介绍几种常用的文件查找命令。

- cd命令：是在Linux文件系统的不同部分之间移动的基本工具。输入cd命令，后面跟着一个路径名，就可以直接进入到另外一个子目录中。
- pwd命令：使用该命令可以确定自己所在的位置，并显示当前工作子目录。
- find命令：是一个功能强大的操作命令，可以使用它在硬盘驱动器上查找文件。
- whereis命令：使用该命令可以迅速地找到文件，而且可以提供这个文件的二进制可执行文件、源代码文件和使用手册页存放的位置。
- locate命令：该命令使用的是一个文件名数据库，检索一个文件要比搜索整个硬盘驱动器要节省时间。

说明

如果想使用man命令查看cd命令的使用手册页，则会发现什么也找不到。因为cd命令是内建在shell中的，而shell在按下回车键之后就直接解释执行命令行。

注意，要想更快、更准地对Linux系统进行搜索查询，就需要将以上几个命令熟练掌握，同时学会联合使用多个命令，这样可以大大提高工作效率。

面试试题2

是否使用过ftp命令进行的数据传输，如果有，请说明ftp命令格式及其使用的指定传输模式的内部命令。

解答

在Linux系统中，ftp命令是互联网用户使用最频繁的命令之一。不论是在DOS还是UNIX操作系统下，都会遇到大量的ftp内部命令，熟练掌握并灵活应用这些内部命令可以大大提高工作效率。

ftp命令格式：ftp –v –d –i –n –g[主机名]。

- －v 显示远程服务器的所有响应信息。
- －d 使用调试方式。
- －n 限制ftp的自动登录，即不使用.netrc文件。
- －g 取消全局文件。

Offer

ftp 中用来指定传输模式的命令是 mode，可以携带参数是 S、B。其中，S 代表 stream 即文件流，B 代表 bin，即使用二进制文件传输方式。

说明

以下 ftp 内部命令未标*的都是已经实现并经常使用的。

- ABOR: 放弃传输。
- *ALLO: 为即将传送的文件分配空间，并通过后面携带的参数来确定字节数。
- *APPE: 将文件附加到已经存在的文件后面。
- CDUP: 在远程系统上将当前目录切换到上级父目录。
- CWD: 改变远程系统的工作目录。
- DELE: 删除远程系统的文件。
- HELP: 读取服务器的帮助信息，如支持的命令的列表。
- LIST: 在一个新建立的数据连接上发送当前工作目录下的文件名列表。
- MKD: 创建目录。
- MODE: 指定传输模式，可携带的参数是 S、B 或 C。
- NLST: 在一个新建立的数据连接上发送一个当前目录下的“完全”的目录列表。
- PASS: 提供一个用户登录密码，必须立即跟随在 USER 命令后。
- PWD: 显示服务器端的当前工作目录名。
- QUIT: 退出登录并终止连接。
- *REST: 从服务器的一个标识处重新开始传输。

面试试题 3

请说出 Linux 系统中用来检查网络连接的几条命令，并对它们分别进行解释。

解答

可以使用多种命令对连接进行检查。主要命令包括：ifconfig、netstate、ping、route。

- ifconfig：通常操作员可使用该命令在网络中配置网络界面也可以使用该命令来查看 PPP 连接的状态信息。
- netstate：是一个检查网络活动、连接、路由表以及其他网络消息和统计数字的命令。
- ping：该命令的主要作用是发送测试数据包并且计算从主机返回应答信息所需要的时间。
- route：通常用来在网络接口中建立和删除网络路由。

面试试题 4

什么是 Linux 环境变量设置？如何定制环境变量？



解答

Linux 是一个多用户的操作系统。多用户意味着每个用户登录系统后都会有自己专用的运行环境，而这个环境又是由一组变量所定义的，这组变量被称为环境变量。用户可以对自己的环境变量进行修改以达到对环境的要求。

环境变量是和 Shell 紧密相关的，它通过 Shell 命令进行设置，同时它又可以被所有当前用户所运行的程序所使用。对于 bash 来说，可以通过变量名来访问相应的环境变量。

以下几条命令可以用来订制环境变量。

- HOME：显示环境变量。
- env：显示所有的环境变量。
- set：显示所有本地定义的 Shell 变量。
- unset：用来清除环境变量。
- readonly：用来设置只读变量。

说明

常见的环境变量。
PATH：决定了 shell 将到哪些目录中寻找命令或程序。
HOME：当前用户的主目录。
HISTSIZE：历史记录数。
LOGNAME：当前用户的登录名。
HOSTNAME：主机的名称。
SHELL：前用户 Shell 类型。
LANGUGE：语言相关的环境变量，多语言可以修改此环境变量。
MAIL：当前用户的邮件存放目录。
PS1：基本提示符，对于 root 用户是#，对于普通用户是$。
PS2：附属提示符，默认是“>”。

面试试题 5

简述进程的启动、终止的方式以及如何进行进程的查看。

解答

在 Linux 中启动一个进程有手工启动和调度启动两种方式。

（1）手工启动：用户输入启动命令，直接启动一个进程的启动方式。可以分为。

- 前台启动：直接在 Shell 中输入命令进行启动。
- 后台启动：启动一个目前并不紧急的进程，如打印进程。

（2）调度启动：系统管理员根据系统资源和进程占用资源的情况，事先进行调度安排，指

Offer

定任务运行的时间和场合，到时系统会自动完成该任务。经常使用的进程调度命令为：at、batch、crontab。

说明

at：在指定时刻执行指定的命令序列。
Batch：要在系统平均载量降到 0.8 以下时执行某项一次性的任务，此时使用 batch 命令。
Crontab：该命令用于安装、删除或列出用于驱动 cron 后台进程的任务表。

4.9 VB/VBScript/JavaScript 语言知识考查

VB（Visual Basic）是由微软公司开发的包含协助开发环境的事件驱动编程语言。它源自于 BASIC 编程语言，拥有图形用户界面（GUI）和快速应用程序开发（RAD）系统，可以轻易地使用 DAO、RDO、ADO 连接数据库，或者轻松地创建 ActiveX 控件。程序员可以轻松地使用 VB 提供地组件快速建立一个应用程序。许多专家把 VB 的出现（1991 年）当做是软件开发史上的一个具有划时代意义的事件，可以说，VB 是第一个“可视”的编程软件。

VB 是一种可视化的、面向对象和采用事件驱动方式的结构化高级程序设计语言，可用于开发 Windows 环境下的各类应用程序。VB 的程序是一种基于窗体的可视化组件安排的联合，并且是通过增加代码来指定组建的属性和方法，其编程机制是以事件驱动。

VBScript 是 VB 的一个子集，是一种解释型语言，由微软公司开发。同其他脚本语言一样，VBScript 被称做一种“胶水”语言，语法简单但功能强大，以实现快速开发。它的语法类似于 VB。VBS 不使用 VB 运行库运行，而是由 Windows 脚本宿主（wscript.exe）解释执行，也可在 IE 浏览器中被解释执行。

JavaScript 是一种广泛用于客户端 Web 开发的脚本语言，常用来为 HTML 网页添加动态功能，例如响应用户的各种操作等。JavaScript 最初由网景公司设计，是一种动态、弱类型、基于原型的语言，内置支持类。虽然更常用于网页，但 JavaScript 也可以用于其他场合，如服务器端编程等。JavaScript 最初的确是受 Java 启发而开始设计的，而且设计的目的之一就是“看上去像 Java”，因此二者语法上有很多类似之处，许多名称和命名规范也源自 Java。但是实际上，JavaScript 的主要设计原则源自 Self 和 Scheme，它与 Java 本质上是不同的。它与 Java 名称上的近似，是当时网景公司为了营销考虑与 Sun 公司达成协议的结果。

JavaScript 是一种脚本语言，其源代码在发往客户端运行之前不需经过编译，而是将文本格式的字符代码发送给浏览器由浏览器解释运行。解释语言的弱点是安全性较差，而且在 JavaScript 中，如果一条运行不了，那么下面的语言也无法运行。另外，由于每次重新加载都会重新解译，加载后，有些代码会延迟至运行时才解译，甚至会多次解译，所以速度较慢。

上述 3 种语言应用较广，在面试中经常会被考查到。



4.9.1 常见考点汇总

- VB 特点、机制、语法及使用（如 ActiveX 控件的创建）。
- VBScript 特点、机制、语法及使用。
- JavaScript 特点、机制、语法及使用。
- Java 与 JavaScript 的区别。
- VB 与 VBScript 的区别。
- VB 与 C 的一些区别。

4.9.2 经典考题及解析

面试试题 1

请在下列有关 VB 语言的论述中选择正确的一项。

A．使用“#”来生成快捷键。

B．在 VB 程序中，每个窗体的文件（.frm）都是自动生成的，生成后可以手工编辑。

C．必须先把 Frame（框架）控件放入窗体中，才能对另外几种控件进行分类。如果先放入这些控件，再放入 Frame，则无法使用 Frame 进行分类。

D．VB 中所有的控件不必都放在窗体中。

解答

答案 C。

分析

- 使用“&”来为后面的字符添加下划线，即生成快捷键。
- 在 VB 程序中，每个窗体的文件（.frm）都是自动生成的，不可以手工编辑。
- VB 中所有的控件都必须放在窗体中。

面试试题 2

请在下列有关 VB 语言的论述中选择正确的一项。

A．VB 程序被启动时，第一个发生的事件是 Resize。

B．按下鼠标左键并抬起，会触发 MouseDown 和 Click 事件。

C．3china 是合法的变量名。

D．VB 只支持两种循环结构，即 Do…Loop、For…Next。

解答

答案 D。

Offer!

分析

- VB 程序被启动时，第一个发生的事件是 Initialize。VB 是事件驱动的，事件一般分为用户事件和系统事件。在应用程序生命周期内，Initialize 一般只触发一次，并且是在窗体 Load 事件之前发生。如果应用程序被正常打开而后正常关闭（未出现 Bug），则发生事件的顺序是：Initialize→Load→Unload→Terminate。Resize 事件在某个对象第一次显示或某个对象的窗口状态改变时（如从最小化到最大化）发生。
- 按下鼠标左键后释放鼠标会触发 MouseDown、Click、MouseUp 事件。
- VB 中的变量名命名规范是。
 - 必须以字母开头。
 - 以字母、数字、下划线组成。
 - 长度不能超过 255 个字符。
 - 在同一范围内（即一个可引用变量的值域：如同一过程、同一窗体）变量名必须唯一。
- VB 只支持两种循环结构：Do…Loop、For…Next。Do…Loop 循环有当型、直到型两种方式。For…Next 语句重复执行指定的次数。循环可以嵌套。

面试试题 3

在 VBScript 中有哪些数据类型？

解答

在 VBScript 中只有一种数据类型：Variant。Variant 变量是一种特殊的变量，它可以依据变量的使用环境存储 VBScript 程序中的各种信息。此外，VBScript 中的函数的返回值也是 Variant。

分析

对于其他语言中常见的变量类型如整型、浮点、字符串等，Variant 可以根据其存储数据的内容进行区分。它甚至可以直接存储时间日期并支持其操作。

VBScript 所提供的 Variant 可存放的数据子类型如表 4-6 所示。

表 4-6 Variant 可存放的数据子类型

子 类 型	描 述
Empty	表示 Variant 未被初始化。作为数字类型（整型、浮点等）变量使用时其值会被自动初始化为零，而作为字符串类型变量使用时其值会被自动初始化为零长度的字符串 ("")
Null	表示 Variant 为存储任何有效数据
Boolean	表示布尔类型，存储 True 或 False
Byte	存储 0 到 255 之间的整数
Integer	存储-32,768 到 32,767 之间的整数
Currency	存储-922,337,203,685,477.5808 到 922,337,203,685,477.5807 之间的整数
Long	存储-2,147,483,648 到 2,147,483,647 之间的整数

Offer!

续表

子 类 型	描 述
Single	存储单精度浮点数。其范围为：负值-3.402823E38 到-1.401298E-45，正值 1.401298E-45 到 3.402823E38
Double	存储单精度浮点数。其范围为：负值-1.79769313486232E308 到-4.94065645841247E-324，正值 4.94065645841247E-324 到 1.79769313486232E308
Date (Time)	存储一个代表日前的数值。其范围为：January 1, 100 至 December 31, 9999
String	存储一个可变长度的字符串。该字符串最长可容纳 20 亿左右的字符
Object	储存一个对象
Error	存储错误返回码

面试试题4

（1）如何在 VBScript 中构造一个类？
（2）如何继承一个 VBScript 的类？
（3）如何在 VBScript 中重载一个函数？

解答

（1）使用 Class 关键字创建一个类。
（2）VBScript 中自定义的类无法被继承。
（3）VBScript 中定义的函数无法被重载。

分析

我们可以使用 Class 关键字在 VBScript 中创建一个自定义类型。该自定义类型可以包括类的数据成员、成员函数及类的属性和初始化函数。

类的数据成员、成员函数及类的属性可以是私有的（Private），也可以是公有的（Public）。当其为私有时，该数据成员、成员函数或类的属性只能为该类内部成员所用；当其为公有时，该数据成员、成员函数或类的属性可以为该类内部和外部成员所用。

VBScript 通过 Class_Initialize()方法提供对构造函数概念的支持。当创建类的新实例时，该方法自动被调用。如果自定义的类包含有 Class_Initialize()方法，对 New 关键字的调用将调用该方法。由于 Class_Initialize()是一个私有方法，且不允许带任何参数，因此，在 VBScript 中创建一个类的实例时，用户无法对其初始化函数传递值。

VBScript 语言不提供内在的对继承、多态和函数重载（多态的另一种表现形式）的支持。下面创建一个类 PerformanceLog：

```
Rem 定义类 PermanceLog
Class PerformanceLog
    Rem 定义数据成员
        Private m_LogName
    Rem 定义类的初始化函数
```

Offer

```
        Private Sub Class_Initialize
            m_LogName = ""
        End Sub
    Rem 定义类的属性
        Rem Property LogName
            Public Property Get LogName
                LogName = m_LogName
            End Property

            Public Property Let LogName(lgnm)
                m_LogName = lgnm
            End Property
    Rem 定义类的成员函数
        Rem function of creating performance log
            Private Function CreatePerfLog()
                ...
            End Function
        Rem function of starting Performance log
            Public Function StartPerfLog()
                ...
            End Function
        Rem function of stopping Performance log
            Public Function StopPerfLog()
                ...
                ...
            End Function
End Class
```

面试试题 5

JavaScript 和 Java 的区别？

解答

Java 是强类型、静态类型语言，而 JavaScript 是弱类型、动态类型语言。

JavaScript 和 Java 在某种程度上看起来很相似，但实际是确有本质的不同。JavaScript 继承了 Java 的语法和基本的控制流结构，但是不包含 Java 的静态类型检查和强类型检查。相对于 Java 的编译时类型检查系统，而 JavaScript 采用的是基于简单类型（数值类型、布尔类型等）和字符串类型的运行时类型检查系统。

与 Java 相比，JavaScript 的语法更自由，使用者甚至可以不声明所有的变量、类和方法，同时使用者也不必关心一个方法是 Public 的、Private 的还是 Protected 的。在 JavaScript 中，变量、参数和函数的返回值的类型都不是显性的，它们的类型都是在运行的时刻才被决定的。

Java 是按快速执行和类型安全的目的设计的。类型安全意味着使用者不可以在 Java 中把一

个简单类型直接转化为一个类对象的引用，也不能越界访问其他对象的私有内存。相比之下JavaScript的解释执行和运行时的类型检查就使JavaScript程序运行起来慢多了。

4.10 HTML/XML知识考查

如今的Internet逐渐深入了我们的日常生活中。成千上万台计算机组成了Internet，如何访问这些海量的信息？1991年访问Internet信息的系统——万维网（World Wide Web）在欧洲物理实验室被发明。万维网相对其他Internet服务的一个重大突破就是HTML语言（超文本标记语言，Hypertext Markup Language）的使用。

HTML是为“网页创建和其他可在网页浏览器中看到的信息”设计的一种标记语言。网站是网页的集合，而网页则又由文本信息、图形、声音、动画和视频文件等元素构成。HTML编写的网页可被IE等浏览器解析，从而使得用户可以访问这些网页。

早期的HTML语法被定义成较松散的规则，浏览器接受了这个现实，可以显示语法不严格的网页。随着时间的流逝，官方标准渐渐趋于严格的语法，但是浏览器仍然继续显示一些远称不上合乎标准的HTML。

XML（eXtensible Markup Language），即可扩展标记语言，于1998年发布为W3C标准（XML1.0）。XML用于传送和携带数据信息，不用于表现或展示数据，而HTML语言则用于表现数据。

使用XML的严格规则的XHTML（eXtensible HyperText Markup Language，可扩展超文本标记语言）是W3C计划中的HTML的接替者，它与HTML类似但更为严格。虽然很多人认为它已经成为当前的HTML标准，但是它实际上是一个独立的、与HTML平行发展的标准。W3C目前的建议是使用XHTML 1.1、 XHTML 1.0或HTML 4.01进行网络发布。

在网站测试工程师的面试笔试中，HTML、XML知识是属于肯定会被考查的领域，下面通过几道典型例题详细进行介绍。

4.10.1 常见考点汇总

- 各种标签的意义、属性及显示结果。
- HTML/XML语言中的规则。
- CSS（层叠样式表）概念及应用。
- XML的语法和规则以及其与HTML的不同之处。
- DTD文件。
- 制作XML网页过程。
- 有效XML与格式良好的XML之间的区别。



4.10.2 经典考题及解析

面试试题 1

请写出以下 HTML 程序运行结果：

```
<DL>
<DT>会计理论<DD>财务报表
<DT>优质企业<DD>个案分析
<DT>借以检视<DD>企业潜质
</DL>
```

解答

会计理论

　　财务报表

优质企业

　　个案分析

借以检视

　　企业潜质

分析

- <DT>表示后面的是列表项。
- <DD>表示后面的是说明文字。
- <DL>与</DL>配对使用，表示二者之间的内容是定义性列表。定义性列表用来给每个列表项加一段说明性文字，说明独立于列表项另起一行显示。

面试试题 2

在下列有关 HTML 语言的论述中，请选出错误的一项。

A．有序列表和无序列表可以互相嵌套。

B．在指定嵌套列表时，可以具体指定项目符号或编号样式。

C．无序列表应使用 UL 和 LI 标记符进行创建。

D．在创建列表时，LI 标记符的结束标记符不可省略。

解答

答案为 D。

- 无序列表使用的一对标签是和。每个列表项前使用。
- 有序列表使用的一对标签是和。每个列表项前使用。



- HTML 的语法并不严格，所以可以省略。

面试试题 3

请使用 HTML 语言实现下面的网页，如图 4-8 所示。

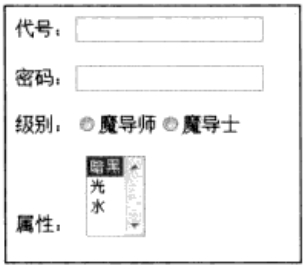


图 4-8 使用 HTML 语言实现的网页

解答

```
<FORM>
  代号：<INPUT><P>
  密码：<INPUT type="password"><P>
  级别：<INPUT type="radio" name="magic">魔导师<INPUT type="radio" name="magic">魔导士<P>
  属性：
<SELECT multiple size=3>
 <OPTION selected>暗黑
 <OPTION>光
 <OPTION >水
</FORM>
```

分析

- <FORM>和</FORM>配对使用，表示其中内容是一个表单。
- <INPUT>标签，代表 HTML 表单的单行输入域。
- type 是<INPUT>标签的一个属性，代表一个输入域的显示方式。radio 表示单选按钮。根据不同的 type 属性值，输入字段拥有很多种形式。输入字段可以是文本字段、复选框、掩码后的文本控件、单选按钮及按钮等。
- name 也是一个属性，用于设置或获取对象的名称（本题中是 magic）。
- <SELECT multiple size=3>，multiple 是 SELECT 的 1 个属性，用于显示列表框。size=3 表示列表框中有 3 个元素。如果此时 size 的值为 4，则列表框中会产生 1 个空值作为第 4 个元素；如果值为 2，则第 3 个元素“水”不会出现在列表框中。
- <OPTION selected>暗黑，OPTION 表示一个列表中的可选项，selected 表示此项是默认选项。

Offer

面试试题 4

XML 文档有哪些规则？

解答

- 必须有 XML 声明语句。
- XML 是大小写敏感的。
- 有效的 XML 文档必须有相应的 DTD 文件，并遵守该 DTD 文件制订的规范。
- 属性值需要加引号（可单引号，也可双引号）。
- 标识必须成对出现，空标识需被关闭（如
）。
- 元素不能交叉。
- 文档中只存在一个称为根的元素或文档元素，该元素无任何部分出现在其他元素中。

面试试题 5

引用 DTD 文件有哪两种方法？

解答

- 方法 1：直接包含在 XML 文档内。在 DOCTYPE 声明中插入说明。
- 方法 2：调用独立的 DTD 文件。将 DTD 文档保存为.dtd 文件，在 DOCTYPE 声明中调用。

4.11 Win32 开发知识考查

Win32 是指支持 32 位计算机的 Windows 环境。在此环境下，多种开发语言都可以调用 API（应用程序接口）函数，实现微软风格的窗口化界面编程。

其中，基于 VC 编译器，以 C++/MFC 或 C/SDK 模式的开发是主要方式。实际上，C 或 C++ 都可以使用 SDK 或 MFC 来开发 Win32 程序，区别在于 SDK 是直接调用 API 函数，而 MFC 一般是通过类的包装后由 MFC 对象来调用 API。

Win32 编程是以窗口为核心，以事件驱动为动力，将程序代码与用户界面分开，开发时在资源文件.rc 描述所需要的资源，编译时资源编译器将所有用户资源集中制作出一个.res 文件，再与程序代码结合在一起，生成可执行的.exe 文件。

在 DOS 下 C 语言程序的入口点是 main 函数，而在 Windows 下程序的入口点是 WinMain() 函数。WinMain()函数必须带有 4 个参数：

int PASCAL WinMain(

- HINSTANCE hInstance,　　//当前实例句柄



- HINSTANCE hPrevInstance, //前一个实例句柄
- LPSTR lpCmdLine, //命令行字符
- int nCmdShow //窗口显示方式

)

在 Windows 应用程序中，窗口过程和消息机制是基础。

Windows 是一个“基于事件的、消息驱动的”操作系统，若在该操作系统下执行一个程序，只要用户进行了影响窗口的动作（如改变窗口大小或移动、单击鼠标等），该动作就会触发一个相应的“事件”。系统每次检测到一个事件时，就会给程序发送一个“消息”，从而使程序可以处理该事件。每个 Windows 应用程序都是基于事件和消息的，而且包含一个主事件循环，它不停地、反复地检测是否有用户事件发生。当每次检测到一个用户事件时，程序就对该事件做出响应，处理完再等待下一个事件的发生。Windows 下的应用程序不断地重复这一过程，直至用户终止程序，用代码来描述就是一个消息处理过程的 while 循环语句。

4.11.1 常见考点汇总

- 应用程序中消息传递与消息处理机制。
- Win32 程序执行过程。
- Windows 体系结构。
- 存储管理、文件管理、作业管理、设备管理。
- 进程与线程。
- 窗口、对话框、菜单等基本控件使用。

4.11.2 经典考题及解析

面试试题1

在 Windows 系统中，关于句柄的说法不正确的是。

A．句柄实质上是一个 32 位的数值。

B．句柄是用来标识系统中的对象。

C．操作系统用句柄来区分是哪一个应用程序。

D．操作系统用句柄来区分是哪一个对象。

解答

答案：C。

分析

一个应用程序可启动多个实例句柄。

句柄实际上是一个数据，是一个 Long（整长型）的数据。

Offer

名柄是一种指向指针。名柄是 WONDOWS 用来标识被应用程序所建立或使用的对象的唯一整数，Windows 使用各种各样的句柄标识诸如应用程序实例，窗口、控制、位图、GDI 对象等。Windows 句柄有点像 C 语言中的文件句柄。句柄是一个标识符，标识对象或者项目。

面试试题 2

下列关于进程与线程关系描述正确的是。

A．线程之间的切换比进程切换所用的时间少。

B．进程可以包含多个线程。

C．线程之间可以通信，比进程之间通信效率要高。

D．线程之间不存在争夺共享资源的问题。

解答

答案：A、B、C。

分析

- 线程之间存在争夺共享资源的问题，此时可用信号量、临界区来解决。进程之间的切换需要内核介入，而同一进程中的多线程由于共享同一地址空间，所以在通信时无须内核介入。
- 线程是进程中的一个执行单元，同一个进程中的各个线程对应于一组 CPU 指令、一组 CPU 寄存器及一堆栈。一个进程至少有一个线程。CPU 为了处理程序，就需要有一些临时存储数据的存储单元，称为寄存器。多任务操作系统下，CPU 必须定时更换任务，这时许多数据就必须放入内存备份。因为寄存器中的内容是不断更换的，很容易（可以说肯定会）丢失数据。此外，一个程序中的跳转如果想返回（如调用函数，最后还要回到主程序中），则参数、原地址都要保存在内存中。一般这些内容都保存于堆栈（一片内存区，有一个栈顶指针，放入数据指针向上移动，读出数据指针下移，指针用 CPU 专门的一个寄存器保存（这里所说的指针，其实就是一个内存地址））中。这样，只要准备 N 个虚拟寄存器和 N 个堆栈就可以有 N 个线程同时运行同一个程序（因为寄存器中还保存有程序当前执行的位置，（虚拟）寄存器不同，位置就可以不同，这 N 个线程就基本互不干扰）。

面试试题 3

下列说法正确的是。

A．进程结构的所有成分都在用户空间内。

B．用户程序能够直接访问进程涉及的数据。

C．线程是内核级的实体。



D．线程结构驻留在用户空间中。

解答

答案：D。

分析

- 进程由代码、数据、内核状态和一组寄存器组成，而线程是由表示程序运行状态的寄存器，如程序计数器、栈指针及堆栈组成。线程不包括进程地址空间中的代码和数据，它是计算过程在某一时刻的状态。进程是一个内核级的实体，其结构的所有成分都在内核空间中，一个用户程序不能直接访问这些数据。线程是一个用户级的实体，其结构驻留在用户空间中，能够被普通的用户级方法直接访问。
- 进程通常被定义为一个正在运行的程序的实例，它由两个部分组成，一个是操作系统用来管理进程的内核对象。内核对象也是系统用来存放关于进程的统计信息的地方。另一个是地址空间，它包含所有的可执行模块或 DLL 模块的代码和数据。另外，它还包含动态分配的空间，如线程堆栈和堆分配空间。每个进程被赋予它自己的虚拟地址空间，当进程中的一个线程正在运行时，该线程可以访问只属于它的进程的内存。属于其他进程的内存则是隐藏的，并不能被正在运行的线程访问。

面试试题4

现在我们开发一个软件测试工程师管理软件，软件中存在一个排序功能。在开发中，按下排序键会发现，尽管程序已对当前所有软件工程师按工资收入高低排序，但是屏幕上工程师并未按顺序显示，只有在刷新时屏幕才变化为正确顺序。请问这是什么原因？

解答

在排序程序部分的最后，本来应当发送重绘消息给主窗口 SendMessage(hwnd,WM_PAINT,1,0) 使之刷新。

面试试题5

在入口函数

```
int WINAPI WinMain(HINSTANCE hInstance,        //当前实例句柄
                   HINSTANCE hPrevInstance,    //前一个实例句柄
                   LPSTR lpCmdLine,            //命令行字符串
                   int nCmdShow                //窗口显示方式
```

中，定义了窗口结构体变量：WNDCLASS wndclass。

那么，请问以下语句各自有何作用？

1．wndclass.style=CS_HREDRAW | CS_VREDRAW;

Offer

2．wndclass.lpfnWndProc=WndProc;
3．wndclass.hIcon=LoadIcon(hInstance,MAKEINTRESOURCE(IDI_ICON1));
4．wndclass.hCursor=LoadCursor(NULL,IDC_ARROW);
5．wndclass.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
6．wndclass.lpszMenuName=MAKEINTRESOURCE(IDR_MAINMENU);
7．f(!RegisterClass(&wndclass)) {MessageBox(NULL,"本程序需要 WINDOWS XP/2000 操作系统!",szAppName,MB_ICONERROR); return 0; }

解答

1~6 是填充窗口类结构。

1．定义窗口类风格，每当窗口横向或纵向大小改变时，窗口实例将被重绘。

2．表明窗口消息处理函数是 WndProc。当 Windows 发送消息给该窗口时，此函数将被调用。

3．加载 Logo。

4．加载光标。

5．使用系统默认画刷。

6．加载主菜单。

7．注册窗口类。

分析

WNDCLASS 结构是这样定义的：

```
typedef struct _WNDCLASS {
    UINT style; //窗口的风格
    WNDPROC lpfnWndProc; //指定窗口的消息处理函数的远指针
    int cbClsExtra; //指定分配给窗口类结构之后的额外字节数
    int cbWndExtra; //指定分配给窗口实例之后的额外字节数
    HANDLE hInstance; //指定窗口过程所对应的实例句柄
    HICON hIcon; //指定窗口的图标
    HCURSOR hCursor; //指定窗口的鼠标
    HBRUSH hbrBackground; //指定窗口的背景画刷
    LPCTSTR lpszMenuName; //窗口的菜单资源名称
    LPCTSTR lpszClassName; //该窗口类的名称
} WNDCLASS;
```

4.12 C#语言知识考查

谈到 C#我们就不得不谈一谈 Java。在 20 世纪 90 年代中后期随着 Java 的出现，人们发现其虚拟机的概念使得应用程序可以不做任何修改就能直接在任何操作系统上使用，而且 Java 所提供

的类库比 Microsoft 提供的 FMC 提供的类库使用起来方便很多。此外 Java 不使用指针，而采用垃圾收集方式，这也降低了开发人员维护内存泄漏的工作量，大大提高了开发人员的工作效率。

鉴于 Java 的诸多优势，微软公司于 2000 年 6 月正式宣布正在开发一种新的技术，其中便包括一种新的编程语言 C#。

C#语言具有一系列特点：

- 面向对象技术：包含大量预建的组建，方便程序员快速开发；提高了程序的重用性。
- 安全性：强类型语言，对各种预定义类型进行越界检查，以防止由于数据溢出所产生的安全问题。
- 多线程：C# 的并发机制，同步共享及处理不同事件，使实时交互容易实现。

C#的常用编译器：Visual Studio。

4.12.1 常见考点汇总

- 值类型和引用类型。
- 装箱和拆箱。
- C#中使用 COM 组建。
- 委托。
- 方法的重载、覆盖的区别。
- 多态。
- 代码安全检测。
- 安全性。
- 垃圾回收机制。
- 多线程与同步、交互、锁。
- 线程与进程关系。
- 线程状态与生命周期。
- 线程安全。
- 对象的序列化。
- 静态方法和静态变量。
- 抽象类及方法。
- 类的继承。
- 接口。
- 公共、私有、匿名、静态间的关系，声明和权限控制。
- 错误与异常处理。
- 值传递和引用传递。
- 运算符和表达式如优先级。
- 条件、循环、多分支语句。



- 递归。
- 数组。
- 文件和流。
- 泛型。
- 集合。

4.12.2 经典考题及解析

面试试题1

结构和类的区别？

C#中结构和类都是从 Object 派生的。不同的是。

（1）结构。

- 结构用来定义值类型。
- 结构不能从其他的类或结构派生，也不能作为其他类或结构的基类。
- 程序员无法自定义结构的默认构造函数，结构的默认构造函数会将所有值类型字段。初始化为零，而将所有引用型字段初始化为空引用。
- 结构的字段不能在声明中显式初始化。
- 因为结构为值类型，所以其实例通常在线程栈中。

（2）类。

- 类用来定义引用类型。
- 类可以从其他的类派生或作为其他类的基类。
- 程序员可以自定义类的默认构造函数。
- 类的字段可以在声明中显式初始化。
- 因为类为引用类型，所以其实例总是在进程堆中。

面试试题2

const 关键字和 readonly 关键字的区别？

const 关键字

const 关键字可以修饰类的字段或局部变量，用于指定变量的值为常数。const 用于修饰编译时常数。

- 常数值是指在编译阶段就可以被完全计算出来的常数表达式或值。

例子

```
class Test
```



```
{
        int varA = 1;
        const int constA = 1;//ok
        const int constB = 1+2;//ok
            const int constC = 1+ constA;//ok
        const int constD = 1 + varA;//error, varA 不是一个常数或 const 修//饰的变量"Error   An object reference is
required for the non-static field, //method, or property 'test.varA'"
        string st = "aaa";
        const string str1 = st; //error, st 不是一个常数或 const 修饰的变量
        const string str1 = "abc";//ok

        readonly int readonlyA = 1; //ok
        public const int constE = readonlyA; // 不能通过编译

        public static void fun()//静态成员函数
        {
            constA = 5;// 不能通过编译，被 const 修饰的类字段一旦被赋值（初始//化），就不可再次被修改
        }
}
```

- 如果被 const 修饰的类字段是引用类型，则字段的值只能是 string 类型常量或 null，不可用其他引用类型的实例进行初始化。

例子

```
class Test
{}
    class TestA
{
    public const string str1 = "abc";//ok

  public const string str2 = null; //ok
        public const Test t1 = null; //ok
        public const Test t2 = new Test();//不能通过编译，不可使用其他引用//类型的实例进行初始化
    }
```

- 被 const 修饰的字段不可使用 static 修饰，因为被 const 修饰的字段自带 static 属性，该字段属于类。

例子

```
class Test
{
    public int varA = 1;
    public const int constA = 1;
    public static void fun()//静态成员函数
    {
        int a = varA; //不能通过编译，varA 是非静态成员
```

Offer

```
            int b = constA;//ok
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            int m = Test.constA;//ok
            int n = Test.varA; //不能通过编译，varA 是非静态成员，不属于类
            Test t = new Test();
            int k = t.constA; //不能通过编译，constA 是静态成员，属于类
            int j = t.varA;//ok
        }
    }
```

- 被 const 修饰的类字段只能在该字段的声明中初始化。

例子

```
class Test
{
    public int varA = 0; //ok
    public const int constA = 1; //ok
    public const int constB = 0; //ok
    public Test()
    {
        varA = 1; //ok
        constB = 1; //不能通过编译，constB 是静态成员，属于类
    }
}
```

- 不可用 const 关键字和 readonly 关键字同时修饰一个变量。

readonly 关键字

readonly 关键字只可用于修饰类的字段，而不可用于修饰局部变量。readonly 用于修饰运行时常数。

例子

```
class Test
{
    readonly int readonlyA = 1;//ok
    public static void fun()
    {
        readonly int readonlyB = 0;// 不能通过编译， readonly 不可用于//修饰局部变量
    }
}
```

- 被 readonly 修饰的类字段只能在该字段声明时或在该类的构造函数中被赋值（初始化）。

Offer

例子

```
class Test
{
    readonly int readonlyA = 1; //ok,在声明时被赋值（初始化）
    readonly int readonlyB; //ok
    public Test()
    {
        readonlyB = 2; //ok，在该类的构造函数中被赋值（初始化）
        readonly int readonlyC;// 不能通过编译， readonly 不可用于修饰局部//变量
    }
}
```

- 被 readonly 修饰的类字段非常量字段进行初始化。

例子

```
class Test
{
    public int varA = 0; //ok
    public const int constA = 0;
    readonly int readonlyA = varA; // 不能通过编译
    readonly int readonlyB = constA; //ok
    readonly int readonlyC = readonlyB; //ok
    public const int constB = readonlyB; // 不能通过编译

}
```

- 如果被 readonly 修饰的类字段是引用类型的字段，其值可用其他引用类型的实例进行初始化。

例子

```
class Test
{}
class TestA
{
  public readonly Test t1 = null; //ok
  public readonly Test t2 = new Test();//ok
  public const Test t3 = new Test();//不能通过编译，不可用其他引用类型的实//例进行初始化。new Test() 所产生的对象不是引用类型常量
    }
```

- 由于被 readonly 修饰的类字段不隐含 static 属性，所以可以用 static 关键字修饰。如果被 readonly 修饰的类字段没有被 static 修饰，则该字段属于对象。

例子

```
class Test
{
```

```
    public const int constA = 1;

    public readonly int readonlyA = 0;

    public static void fun()//静态成员函数
    {
        int b = constA;//ok
        int c = readonlyA;//不能通过编译，readonlyA 是非静态成员
    }
}
class Program
{
    static void Main(string[] args)
    {
        int m = Test.constA;//ok
        int n = Test.readonlyA; //不能通过编译，readonlyA 是非静态成员，不//属于类
        Test t = new Test();
        int k = t.constA; //不能通过编译，constA 是静态成员，属于类
        int j = t.readonlyA;//ok
    }
}
```

- 不可用 const 关键字和 readonly 关键字同时修饰一个变量。

面试试题 3

值类型和引用类型的区别？

总体来说，值类型和引用类型都是从 Object 类派生而来的。值类型变量存储的是该类型的值，而引用类型变量存储的是该类型变量的实例（或称对象）的内存地址。下面介绍一下它们的区别：

值类型

- 值类型包括 13 种基本类型（SByte、Byte、Int16、UInt16、Int32、UInt32、Int64、UInt64、Char、Boolean、Double、Single、Decimal）、enum 和 struc。
- 值类型实例内存分配属于静态分配。
- 值类型实例通常分配在线程的栈上，但在有些情况下可以存储在堆中。

例子

例如自定义类中的某个值类型字段，就会与该类的实例一同放在堆中。

- 值类型的实例没有引用。
- 在 C#中可以用 NEW 来分配值类型实例。



例子

例如初始化一个结构。

- 值类型实例作为函数参数或返回值时，传递的是该实例的一个拷贝。

引用类型。

- 除 13 种基本类型以及 enum 和 struct 之外的所有类型。
- 引用类型实例内存分配属于动态分配。
- 引用类型实例总是分配在进程的堆上。

例子

例如，当一个结构中包含有一个引用类型的字段时，即使该结构的实例定义在栈上，其引用字段的实例还是被分配在栈上，而该引用字段的引用被存储在栈上。

- 所有的引用类型都是类或接口。
- 引用类型实例作为函数参数或返回值时，传递的是该实例的地址。

面试试题 4

什么叫装箱和拆箱？

为什么会有装箱和拆箱，有以下 3 个原因。

- 值类型和引用类型都是从 Object 类派生而来的，Object 的引用可引用其任何派生类。
- 值类型的实例没有引用。
- 引用类型实例总是分配在进程的堆上。

例子

```
Class ShowBox                                                    1
{                                                                2
    static void doBox(object o)//参数是 Object 类型的一个引用    3
    {...}                                                        4
    public static void Main(string[] Args)                      5
    {                                                            6
        int i = 0;                                               7
        doBox (i);//将值类型作为参数传递给一个引用类型            8
    }                                                            9
}                                                                10
```

在上述例题中，函数 doBox 的形参是一个 Object 类型的引用（第 3 行）。而主函数中第 8 行将一个值类型 i 作为实参直接传递给了函数 doBox。此处 doBox 函数的形参 o 应当可以接收值类型 i。因为它是 Object 的引用，而值类型是 Object 类派生而来的。Object 类的引用指向其派生类的实例是理所当然的。

Offer!

但是作为一个引用类型，形参 o 必须指向一个分配在进程堆上的引用类型实例的地址。而 i 的实例是分配在线程栈上的一个值类型实例，因此它没有引用。这样，doBox(object o)应当如何如何传递参数呢？于是 C#中就有了装箱的操作，注意，该操作是隐式执行的。

装箱操作步骤如下。

（1）在进程堆中建立一个新的对应于线程栈中被装箱的值类型实例（该例中 i 的类型 int）的引用类型实例。

（2）用线程栈中被装箱的值类型实例的值对进程堆中新建立引用类型实例进行初始化。

（3）将 Object 的引用 o 指向进程堆中新建立引用类型实例。

装箱的反向操作被称作拆箱，这个操作也是隐式执行的。

例子

```
public static void Main(string[] Args)
{
    char c = 'a';
    object o= c;//装箱
    char t = (char)o; //拆箱
}
```

此例中，char t = (char)o 语句将存放于进程堆中的 object 引用类型的值通过拆箱操作拷贝给了分配在线程栈上的值类型变量 t。

如果被拆箱的引用为空，则系统会抛出 NullReferenceException 异常。

面试试题 5

什么是委托？

C++中有函数指针的概念，而 C#中则是通过使用委托来实现与 C++中函数指针相似的功能，而且委托的功能比函数指针更为强大。它可以引用一个或多个方法,只要这些方法的签名和该委托的签名相同即可。

当一个委托引用了多个方法时，我们调用这个委托就相当于调用了其引用的所有方法。这样的调用方式称作多点广播（multicast）。

C#中的委托类必须用 delegate 关键字来创建（类似于普通类型必须用 class 关键字来创建）。

例子

```
delegate void delegt1();
 //用 delegate 关键字申明一个委托类 delegt1，这个代理类的签名是
 //void fun()
delegate int delegt2(int i);
```

Offer!

```
//用 delegate 关键字申明一个委托类 delegt2，这个代理类的签名是
//int fun(int)
```

被创建出来的委托类就可以用来创建委托对象。

例子

```
delegt1 delegInstance1 = new delegt1(fun1);
//函数 fun1 必须有和委托类 delegt1 相同的签名 void fun()
delegt2 delegInstance2 = new delegt2(fun2);
//函数 fun2 必须有和委托类 delegt2 相同的签名 int fun(int)
```

以上委托对象的创建等价于：

```
delegt1 delegInstance1 = fun1;
//函数 fun1 必须有和委托类 delegt1 相同的签名 void fun()
delegt2 delegInstance2 = fun2;
//函数 fun2 必须有和委托类 delegt2 相同的签名 int fun(int)
```

这是一种隐式创建委托对象的方式，在创建中调用了委托类 delegt1 和 delegt2 的构造函数。

一个委托引用多个函数：

```
delegt1 delegInstanceA = funA;
delegInstanceA += funB;
delegInstanceA += funC;
//函数 funA, funB, funC 必须有和委托类 delegt1 相同的签名 void fun()
delegInstanceA();
//调用 delegInstanA()将使函数 funA, funB, funC 都被调用
```

⚠ 注意

```
delegt1 delegInstanceB = funB;
delegInstanceB += funB;
//函数 funB 被委托对象 delegInstanceB 引用两次
delegInstanceA();
//调用 delegInstanA()将使函数 funB 被调用两次
```

从一个委托中减去对函数的引用：

```
delegt1 delegInstanceC = funC;
delegInstanceC -= funC;
//委托对象 delegInstanceB 失去对函数 funC 的引用
delegInstanceA();
//调用 delegInstanA()将不会调用函数 funC
delegInstanceC -= funC;
//再次从 delegInstanceB 减去对函数 funC 的引用并不会引起编译器的报错
```

委托的常见用法有用委托来连接控件的事件和其响应方法，以及用委托来传递参数以实现泛型结构中的排序等操作。

Offer

4.13 数据库知识考查

简单来说，数据库就是信息的集合，也就是用于存放数据的库。通过数据库可以很方便快捷地管理查询数据。严格地说，数据库是“按照数据结构来组织、存储和管理数据的仓库”，是依照某种数据模型组织起来并存放二级存储器中数据的集合。这种数据集合具有如下特点：尽可能不重复，以最优方式为某个特定组织的多种应用服务，其数据结构独立于使用它的应用程序，对数据的增、删、改和检索由统一软件进行管理和控制。从发展的历史看，数据库是数据管理的高级阶段，它是由文件管理系统发展起来的。

数据库系统从基本上讲是计算机化的数据保存系统，它主要以计算机化的方式来维护数据且让用户依据不同的需求快速取得信息。相较于数据库系统，数据库是一个比较抽象化的概念，凡是相关数据的集合都可以称为数据库，而数据库系统则是一个较具体的概念，是计算机化作业的数据库。

数据库存在多种模型，应用于大型数据储存的数据库一般为网状数据库（Network Database）、关系数据库（Relational Database）以及面向对象式数据库（Object-Oriented Database，Oracle 即拥有此特性）。此外，也有应用在 LDAP（轻量级数据访问协议）的层次结构式数据库（Hierarchical Database）。

4.13.1 常见考点汇总

- 数据库基础知识。
- 系统数据库。
- 了解文件和文件组。
- 事务日志。
- 数据库及文件状态。
- 数据库的设计。
- 数据的完整性。
- 设计文件和文件组。
- 创建数据库。
- 修改数据库，添加和删除数据文件。
- 数据库的查询。
- 表的概念及类型。
- 对表的操作。
- 创建、查询、更新、删除表。
- 设计实现索引。



- 表的主键外键。
- 设计和实现空间存储。
- SQL Server、Oracle 等具体数据库的了解。

4.13.2 经典考题及解析

面试试题1

请解释一下数据库设计中的5个范式？

解答

- 第一范式：对于表中的每一行，必须且仅仅有唯一的行值。在一行中的每一列仅有唯一的值且具有原子性。第一范式是通过把重复的组放到每个独立的表中，再把这些表通过一对多关联联系起来的这种方式来消除重复组。
- 第二范式：第二范式要求非主键列是主键的子集，非主键列活动必须完全依赖整个主键。主键必须有唯一性的元素，一个主键可以由一个或更多的组成唯一值的列组成。一旦创建，主键将无法改变，外键关联一个表的主键。主外键关联意味着一对多的关系。第二范式处理冗余数据的删除问题。当某张表中的信息依赖于该表中其他的不是主键部分的列时，通常会违反第二范式。
- 第三范式：第三范式要求非主键列互不依赖。第三范式规则查找以消除没有直接依赖于第一范式和第二范式形成的表的主键的属性。我们为没有与表的主键关联的所有信息建立了一张新表。每张新表保存了来自源表的信息和它们所依赖的主键。
- 第四范式：第四范式禁止主键列和非主键列一对多关系不受约束。
- 第五范式：第五范式将表分割成尽可能小的块，目的是为了排除在表中所有的冗余。

说明

在进行数据库设计时，大家应该时刻地注意上述几个范式。一般数据库的设计只需满足3 NF即可。尽管第五范式最难实现的，但用户还是需要在使数据库的设计满足需求的基础上尽量地实现这些功能。

面试试题2

简述数据库系统的三级模式结构。

解答

数据库系统结构是数据库系统的一个总的框架。数据库系统的三级模式结构是指数据库系统是由模式、外模式和内模式三级构成的。



（1）模式：模式也称逻辑模式或概念模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。模式实际上是数据库数据在逻辑级上的视图。一个数据库只有一个模式。定义模式时不仅要定义数据的逻辑结构，而且要定义数据之间的联系，定义与数据有关的安全性、完整性要求。

（2）外模式：外模式也称用户模式，它是数据库用户能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。外模式通常是模式的子集。一个数据库可以有多个外模式。应用程序都是和外模式打交道的。外模式是保证数据库安全性的一个有力措施。每个用户只能看见和访问所对应的外模式中的数据，而数据库中的其余数据对用户来说是不可见的。

（3）内模式：内模式也称存储模式，一个数据库只有一个内模式。它是数据物理结构和存储方式的描述，是数据在数据库内部的表示方式。

面试试题3

数据库的特点有哪些，请分别解释。

解答

（1）实现数据共享。

数据共享既包括所有用户可同时存取数据库中的数据，也包括用户可以用各种方式通过接口使用数据库，并提供数据共享。

（2）减少数据的冗余度。

与文件系统相比，由于数据库实现了数据共享，从而避免了用户各自建立应用文件，这样就减少了大量重复数据，减少了数据冗余，维护了数据的一致性。

（3）数据的独立性。

数据的独立性既包括数据库中数据库的逻辑结构和应用程序相互独立，也包括数据物理结构的变化不影响数据的逻辑结构。

（4）数据实现集中控制。

在文件管理方式中，数据处于一种分散的状态，不同的用户或同一用户在不同处理中其文件之间毫无关系。利用数据库可对数据进行集中控制和管理，并通过数据模型表示各种数据的组织以及数据间的联系。

（5）数据一致性和可维护性，以确保数据的安全性和可靠性。

- 安全性控制：以防止数据丢失、错误更新和越权使用。
- 完整性控制：保证数据的正确性、有效性和相容性。
- 并发控制：使在同一时间周期内，既允许对数据实现多路存取，又能防止用户之间的不正常交互作用。
- 故障的发现和恢复：数据库管理系统提供了一套对于故障恢复和发现的机制。

Offer

（6）故障恢复。

由数据库管理系统提供一套方法，可及时发现故障和修复故障，从而防止数据被破坏。数据库系统能尽快恢复数据库系统运行时出现的故障，可能是物理上或是逻辑上的错误，例如对系统的误操作造成的数据错误等。

面试试题4

请分别解释 INNER JOIN、OUTER JOIN、LEFT JOIN、RIGHT JOIN、FULL JOIN?

解答

- INNER JOIN：内联接的结果集是所要连接的两个表的公共部分数据值。
- OUTER JOIN：外联接包括左向外联接、右向外联接或完整外部联接，也就是下面讲到的3个概念。
- LEFT JOIN：左向外联接的结果集包括 LEFT OUTER 子句中指定的左表的所有行，而不仅仅是联接列所要匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值。
- RIGHT JOIN：右向外联接是左向外联接的反向连接，它将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值。
- FULL JOIN：完整外部联接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。

面试试题5

请解释 Stored Procedure（即存储过程）的含义及用途。

解答

存储过程（Stored Procedure）是一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中。用户通过指定存储过程的名称并给出参数（如果该存储过程带有参数）来执行它。

在 SQL Server 系列版本中存储过程一般分为两类：系统提供的存储过程和用户自定义的存储过程。系统提供的存储过程主要存储在 master 数据库中并以 sp_为前缀，其主要是从系统表中获取信息，从而为系统管理员管理 SQL Server 提供支持。通过系统提供的存储过程，MS SQL Server 中的许多管理性或信息性的活动（如了解数据库对象、数据库信息）都可以被顺利有效地完成。用户自定义的存储过程是由用户创建并能完成某一特定功能（如查询用户所需数据信息）的存储过程。

说明

在 SQL Server 中通常应用 Transaction-SQL 编写程序。存储过程允许标准组件式编程。

Offer

存储过程在被创建以后可以在程序中被多次调用，而不必重新编写该存储过程的SQL语句，而且数据库专业人员可随时对存储过程进行修改，但对应用程序源代码毫无影响（因为应用程序源代码只包含存储过程的调用语句），从而极大地提高了程序的可移植性。

4.14 主流测试工具知识考查

测试工具在软件测试中至关重要，若使用得当，可以大大节省人工。功能或性能自动化工具没有上、下班的限制，大都可以胜任在无人值守状态下运行。自动化工具可以节省大量运营成本。另外，有的测试比如性能测试只能通过自动化工具来进行，根本无法用手工的方式完成。

由于测试工具如此重要，所以使得测试工具尤其是功能/性能自动化测试知识考查在测试工程师的面试中不可或缺。

例子

例如Mercury Interactive公司（2006年被HP公司收购）的性能测试工具LoadRunner，基本上每次面试都是必问项。

测试工具有两大类，一类是产品工具，不针对特定项目，普适性强，但专门性差，例如以上提到的性能测试工具LoadRunner；另一类是项目工具，针对特定项目而开发的工具，没有普适性，例如摩托罗拉公司的手机功能测试工具FlexAuto。

本节主要讨论产品工具的使用，因为其应用面较广。至于项目工具，我们将其归属到项目经验考查范畴，本节不涉及项目工具的考查。

警示

项目工具虽然使用面窄且用户少，但面试官也可能会感兴趣。面试官希望通过项目工具的询问，考查应聘者的专业知识。关于考查要点及举例，可参阅本书第2.3小节、第7章以及第8.4小节的相关内容。由于面试官不太可能熟悉这种特定的项目自动化工具，所以应聘者在此局部领域完全有可能会保持心理优势。此时建议应聘者多谈一些深层次的知识，使面试官感觉到你深厚的底蕴。那些你懂而面试官不懂的地方，很可能就是你得以录用的关键所在。

性能或功能自动化测试的步骤如下。

（1）计划测试场景。

（2）录制测试过程成为自动化测试脚本。

（3）增强和改进录制的自动化测试脚本。

（4）执行自动化测试脚本完成自动化测试。

（5）分析测试结果。

以下几种测试工具需要重点掌握：

Offer

（1）Quick Test Professional（QTP），是Mercury Interactive公司推出的一款功能自动化测试工具，它是专门针对B/S模式的测试工具，是WinRunner的换代产品。该测试工具具有识别能力强、回放精确等优点。

（2）WinRunner，是Mercury Interactive公司早期的功能测试工具，用于检测应用程序是否能够达到预期的功能及正常运行。通过自动录制、检测和回放用户的应用操作，WinRunner能够有效地帮助测试人员对复杂的企业级应用的不同发布版进行测试，提高测试人员的工作效率和质量，同时确保跨平台的、复杂的企业级应用无故障发布及长期稳定运行。

（3）LoadRunner，是Mercury Interactive公司的一种性能测试工具，它通过模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题。LoadRunner是一种适用于各种体系架构的自动负载测试工具，支持广泛的协议和技术。

（4）TestDirector(TD)，是Mercury Interactive公司开发的基于Web的测试管理系统，它可以在公司内部或外部进行全球范围内测试的管理，并通过在一个整体的应用系统中集成了测试管理的各个部分，包括需求管理、测试计划、测试执行以及错误跟踪等功能。与之类似功能的软件有不少，例如著名的开源工具Bugzilla。很多大公司有自己的测试管理系统（例如摩托罗拉Linux-Java手机部门使用DDTS，汤森路透北京研发中心媒体部使用Scarab）。

（5）Jmeter，开源工具，用于对静态的和动态的资源（如文件、Servlet、Perl脚本，Java对象、数据库和查询以及FTP服务器等）的性能进行测试。它可以用于对服务器、网络或对象模拟繁重的负载来测试它们的强度或分析不同压力类型下的整体性能。另外，还有一些性能测试工具如OpenSTA、WebLOAD等也需要有所了解。

（6）Selenium，是一个开源的基于Web的测试工具，采用JavaScript来管理整个测试过程，包括读入测试套件、执行测试和记录测试结果。该测试工具采用JavaScript单元测试工具JUnit为核心，模拟真实的用户操作，包括浏览页面、单击链接、输入文字及提交表单等，并且能够对结果页面进行种种验证。Selenium套件包括：

- Selenium Core;
- Selenium IDE;
- Selenium RC。

另外，还有一些安全测试的工具（如微软Fortify）、单元测试工具（如xUnit系列）及Web测试工具Rational Robot等需要有一定的了解，当然有使用经验是最好。同时，还有些小工具如键接测试工具Xenu等可能在面试中被提及。

下面，我们给出一些典型的例题，以供大家参考。

4.14.1 常见考点汇总

- 性能测试工具LoadRunner、Jmeter的使用、术语、结果分析。
- 功能测试工具QTP的使用、术语。
- 功能测试工具Winrunner的使用、术语。



- 测试管理系统 TD、Bugzilla 的使用、术语。
- Web 功能测试工具 Selenium 的使用、术语。
- 单元测试工具 xUnit 系列的使用、术语。

4.14.2 经典考题及解析

面试试题 1

LoadRunner 中的集合点有何意义?

解答

使机器虚拟出的多用户可以同一时间以并发方式执行任务，更接近真实情况下用户并发的情况。

分析

例如，在 Web 测试中测试并发用户访问时，需要使 HTTP 请求实现“并发”，即同时访问服务器。当线程一个一个启动时是不能实现这个要求的，因此 LoadRunner 采用了“集合点”这一方法。顾名思义，就是先启动的进程等候后启动的，然后在某个时间点一起发送请求。

LoadRunner 提供了以下 3 种集合方式（policy）。

（1）当所有的虚拟用户数满足设定的百分比时就不再等待，释放虚拟用户。

（2）当运行中的虚拟用户数满足设定的百分比时就不再等待，释放虚拟用户。

（3）到虚拟用户数达到某个设定的具体值时就不再等待，释放虚拟用户。

插入集合点如图 4-9 所示。



图 4-9 插入集合点

面试试题 2

LoadRunner 中的关联是什么意思?

解答

将服务器发送的某次请求的动态 ID 信息，利用函数加以处理，使发送方再次发送相关请求时，可以发送相关联的这个已变化的动态信息。

Offer

- 这是笔者曾在面试中碰到的一道真实考题。关联是 LoadRunner 中非常重要的基础概念，常被考到。
- 在测试中，VuGen 模拟自身为浏览器，试图欺骗服务器，而有些智能的服务器则不易被欺骗。这些服务器可能在每个浏览器第一次请求数据时，在数据中放入唯一标识码（Session ID）。如果多个 VuGen 发出的请求被发现是同样的 Session ID，则服务器就认为是无效请求。因此，我们需要找出这个 Session ID，并进行参数化替换，这样就可以每次都产生不同的 Session ID，进而可以成功欺骗服务器。
- VuGen 提供了自动关联和手动关联两种关联方式。

内建关联如图 4-10 所示。

面试试题3

使用 Jmeter 测试 B-S 结构软件，当进行大数据量测试时，随着用户数量的不断增加，服务器吞吐量如图 4-11 所示，请根据该图对该软件进行性能分析。

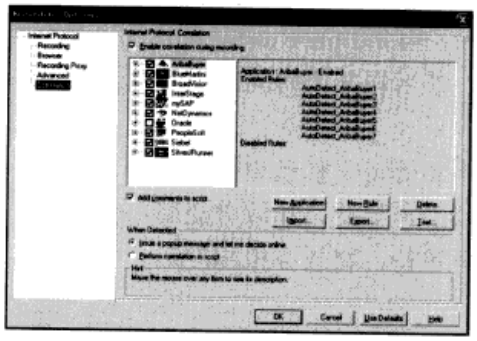


图 4-10 内建关联

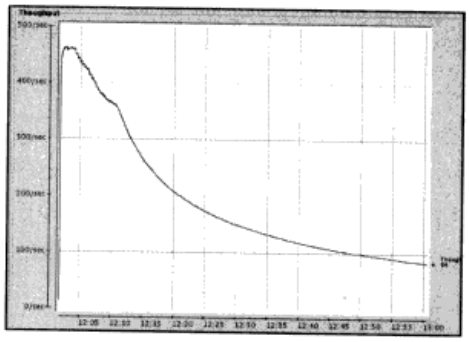


图 4-11 服务器吞吐量随时间变化图

解答

Jmeter 记录的吞吐量（Throughput）是网卡发送与接收的数据量之和（LoadRunner 记录的吞吐量是接收到服务器返回的所有字节数之和）。吞吐量通常与 TPS（服务器每秒处理交易数）和每秒点击率成正比。

在图 4-11 中，吞吐量在到达一定程度后慢慢降低直到接近于零，这反映了系统性能趋于崩溃。显然，此时流量超过系统能处理的最大流量。有很多情况会导致崩溃，如内存耗尽等。瓶颈既可能存在于服务器环节，也可能存在于网络环节。

到底存在于哪个环节，首先要对比网络响应时间和服务器响应时间哪个更费时，然后再分析更细小组件的时间，查看时间到底耗费在哪个环节。同时，还要关注每一步执行的错误率。如果错误率较高甚至达到 100%，即使这一步时间耗费少，也需要探求原因。

如果瓶颈发生在服务器环节的一个或几个组件上，则定位瓶颈原因应按照以下优先级排列。

（1）检查被测软件程序。

Offer

（2）检查 Web 服务器。

（3）检查数据库服务器。

（4）检查操作系统。

分析

- 服务器性能正常情况下吞吐量随时间变化图如图 4-12 所示。

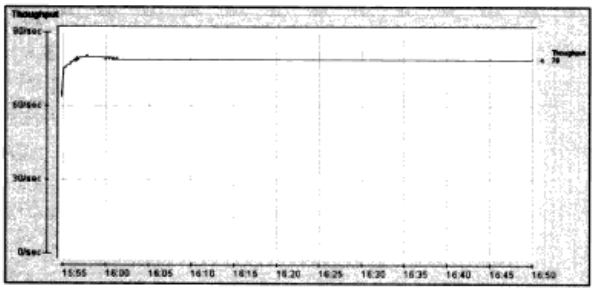


图 4-12 服务器性能正常情况下吞吐量随时间变化图

如果随着用户数量的增加，而吞吐量持平，则说明此时系统吞吐量达到了最大值，系统达到了瓶颈。

实际上，有多种情况都会导致出现如图 4-13 所示的变化图，下面分别列出。

- 数据库连接数达到最大值。
- 网络服务器连接数达到最大值。
- 应用服务器连接数达到最大值。
- CPU 占用达到 100%。
- 在测试过程中用户数量不变（此用户数量远远没有使系统达到瓶颈）。
- 网络带宽达到瓶颈。
- Cache 机制。

从图 4-13 中可以判断出内存没有被耗尽。若被耗尽，则性能会趋向崩溃，应当出现如图 4-12 所示的变化图。

面试试题 4

什么是 Rational Robot?

解答

Rational Robot 是一款用于回归测试的录制回放自动化测试工具，它广泛支持 Java、Web 和.NET 等多种 UI 控件。

Rational Robot 的功能如下。

- 录制和回放测试人员对 GUI 界面的操作，并将操作顺序以脚本形式进行保存。

Offer!

- 检测 GUI 空间在检测点处的状态。
- 可与其他 Rational 工具配合进行其他测试和测试项目管理，例如与 TEST Manager 配合做性能测试等。
- 可在 SQA、VB、VU 环境中编写脚本。
- 可测试 Visual Basic、Oracle Forms、Java、HTML 等软件开发的应用程序。

表 4-7 所示为用于功能测试的 GUI 脚本和用于性能测试的 VU 脚本的区别。

表 4-7 用于功能测试的 GUI 脚本和用于性能测试的 VU 脚本的区别

	GUI	VU
执行原理	记录对 GUI 对象发送的键盘击打或鼠标点击事件	客户端向服务器发送请求
适用测试种类	主要是功能测试，也可执行性能测试	主要用于性能测试（如用户负载测试）
验证点	可用于验证点检查	不支持验证点检查
并发性	一台测试计算机上只能执行一个脚本用例	一台测试计算机上可执行多个脚本用例
脚本语言	SQA Basic	VU
执行环境	既可在 Robot 中执行，又可在 Test Manager 中执行	在 Test Manager 中执行

面试试题 5

如何使用 Windows 自带的 Performance 工具监测服务器或应用程序的性能？

解答

下面以英文版 Windows 2003SE 为例，描述如何通过 Windows 自带的 Performance 工具创建一个性能检测记录器，具体步骤如下。

（1）通过选择 Start→Control Panel→Administrative Tools→Performance 命令启动 Performance 窗口。

（2）在 Performance 窗口中新建一个性能计数器，如图 4-13 所示。

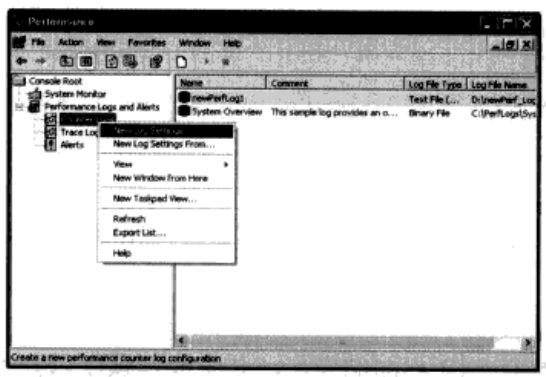


图 4-13 新建一个性能计数器

Offer

（3）向性能检测记录器中添加监测对象或监测算子，如图 4-14 所示。

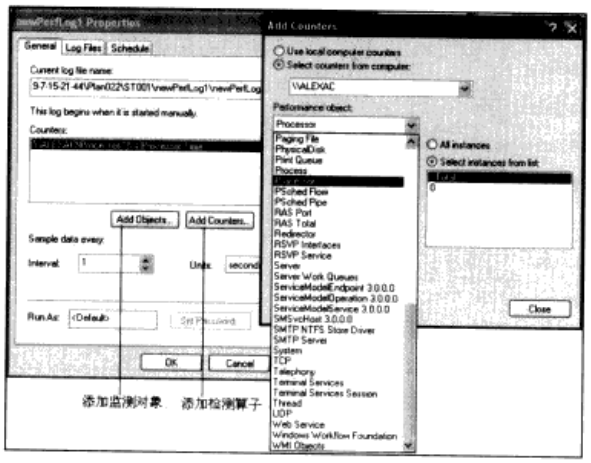


图 4-14 向性能检测记录器中添加监测对象或监测算子

以添加监测算子为例，我们可以监测整个系统的 CPU 等系统资源的使用情况，同时也可以监测单个进程的资源使用情况。图 4-15 所示为监测正在运行的 notepad 进程的 Virtual Bytes 的使用情况。

（4）其他参数设置，如下所示。

- Interval：采样点时间间隔。
- Log Files：可设置日志文件的命名方式和路径。
- Schedule：设置计划任务。

（5）启动和停止性能计数器，如图 4-16 所示。

（6）查看性能监测日志。监测日志通常是 CSV 文件，我们可以使用 Excel 等软件进行查看，并对采用数据进行绘图操作。

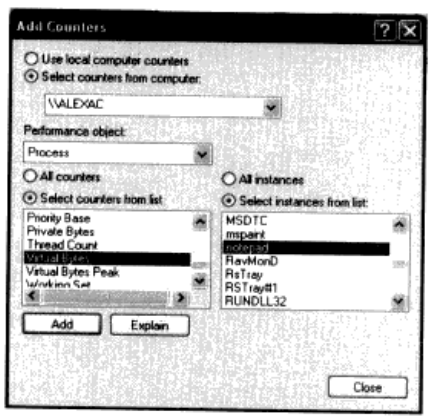


图 4-15 监测正在运行的 notepad 进程的 Virtual Bytes 的使用情况

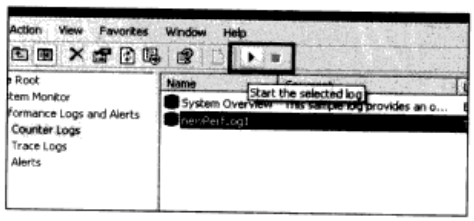


图 4-16 启动和停止性能计数器

如图 4-17 所示就是通过一个性能计数器纪录下来的某个应用程序 Private Bytes 的使用情况。

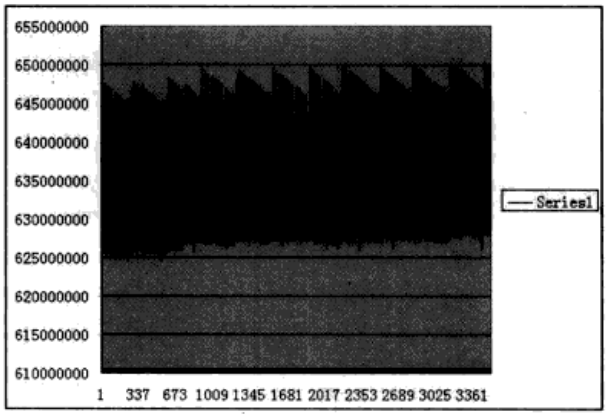


图 4-17 通过一个性能计数器纪录下来的某个应用程序 Private Bytes 的使用情况

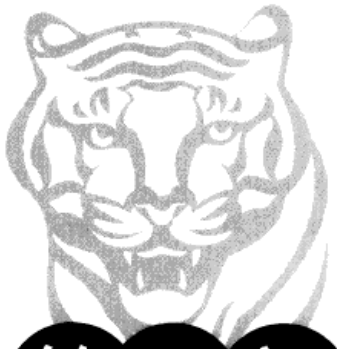
Windows 提供了 Performance 工具的命令行支持，以支持脚本自动化。同时，其还提供了 Logman 命令，用于在本地和远程系统上管理和计划性能计数器与事件跟踪日志的收集。语法如下：

Logman [create {counter | trace} [CollectionName]] [start CollectionName] [stop CollectionName] [delete CollectionName] [query {CollectionName | providers [providerName]}] [update CollectionName]

关于 Logman 命令的详细信息，我们也可以参考：

http://technet.microsoft.com/zh-cn/cc755366(WS.10).aspx

Offer



第三部分

测试开发篇

在第二部分测试理论篇，通过对软件测试基础概念和原理以及相关计算机应用两大部分内容的介绍，大家对软件测试的面试笔试考查重点和方式又多了一层认识。扎实的理论基础至关重要。软件测试工程师需要涉及多种计算机应用领域，“知识面广”是其特点。

在本篇是提高篇，将主要讲解测试开发。这是本书中最难的一篇。欠缺计算机基础知识和开发经验的读者，可能会有些吃力。如果读者能将本篇的内容完全掌握并且融会贯通，得到 1 万元以上月薪的测试开发工作是轻而易举的。在本篇，我们将结合测试开发的知识，比照公司招聘考查的要求，给出一些面试笔试题目及解析。

所谓测试开发，就是指软件测试中的开发。这种开发通常分为：测试工具的开发和测试用例的开发。其中，测试工具的开发更难、层次更高。

测试工具的开发分为产品和项目两种用途。例如著名的性能测试工具 LoadRunner 就是一种产品，在市场上得到广泛应用。开发这类测试工具的工程师，与其说是测试工程师，还不如说是开发工程师。因此，我们这里讲的测试开发，主要是指应用于特定项目的测试工具开发。

从另一方面来讲，应用广泛往往意味着独特性差。很多测试项目有自己的特殊需求，即使公司上层愿意花钱，市场上也没有现成软件符合这些特殊需求。这时，我们就需要自己进行测试开发。

测试一个软件，时间应该不会很长。如果使用很多时间来开发测试工具，就得不偿失。因此要开发的工具往往不是很复杂的工具，功能和界面的要求都要求实用、简便、高效。当然，一般情况下在开发中也不会只使用于一个项目的工具，领导在考虑开发此工具时，是希望可以复用于以后项目的。

测试用例的开发，通常是使用某种特定的脚本语言编写程序（一步步实现测试用例），最后在测试工具上运行。测试工具可以大大提高效率、代替人工（如无人值守时亦可工作）以及避免马虎大意等失误。

思索

但是自动化的测试用例往往会漏掉一些 Bug，因为错误可能千奇百怪，事先无法预料。人工检测可以发现这些缺陷，而机器只是按部就班，难免顾此失彼。因此很多时候，自动化测试通过的测试用例也不一定没有问题。但若是测试用例考虑到过多的错误可能性兼容并包，会大大延缓测试的速度，修改脚本时也比较麻烦。因此，建议测试用例还是以简洁为上，在执行时，一遍自动化测试配合一遍手工测试，这样就能减缓风险，同时也比纯手工效率要高。

本篇涉及的主要内容是用于特定项目的小型测试框架的搭建和使用以及测试工具的开发，主要使用的开发语言是 Perl 和 Python。在小型测试工具的开发中，这两种语言应用得非常广。

目前市面上讲解 Perl 和 Python 测试相关的书籍非常少，尤其是 Python。因此，对 Perl 和 Python 感兴趣的读者来说，本篇也是很不错的 Perl 和 Python 的入门及进阶讲解。

Perl 是一种解释运行的语言，其设计原则为实用第一，并引入了模块的设计思想。Perl 提供和超越很多脚本语言（如 Sed 和 Awk）的所有功能，它很容易操作数字、文本、文件与目录、计算机和网络，特别是程序，同时也很容易运行外部的程序并且扫描这些程序的输出获取感兴趣的东西，还能把这些东西交给其他程序做特殊的处理，另外，它还很容易在任何现代的操作系统上编译和运行。

Python 是一种面向对象的脚本编程语言，这种脚本语言特别强调开发速度和代码的清晰程度，从简单的脚本任务到复杂的面向对象的应用程序都可应用。Python 的扩展性较强同时执行严格的编码标准，同时，它具有数量众多的模块和库。JPython 是用 Java 写的 Python，它完全支持 Java，在这个环境下使用 Python 可以随意地使用 Java 的类库。

下面，请大家跟随我进入测试开发的空间。

第5章 软件测试面试 Perl语言考查

5.1 Perl语言考查

PERL是Practical Extraction and Report Language的简写,标准全称为使用摘要和报表语言。它是由Larry Wall设计的，并由他不断更新和维护的用于在UNIX环境下的一门解释性语言。虽然Perl是为UNIX操作系统设计的，但是Perl以其可移植性和捆绑在大多数操作系统上而闻名，具有广泛的应用。由于Perl使编程更加简单和灵活快速，因此该语言深受广大使用者青睐，不止是专业的程序员还包括多个行业的技术人员，用户的数量也在不断飞速增长。

Perl继承了UNIX的很多优点，其脚本在功能上类似于UNIX awk、sed、shell脚本和C程序。shell脚本主要由UNIX命令构成，而Perl脚本则不是。awk和sed用于编辑和报告文件，但Perl无须执行文件就能工作。C没有与shell、awk和sed的模式匹配和通配符，而Perl却有扩展的字符集。简而言之，Perl像C一样强大，像awk、sed等脚本描述语言一样方便。Perl最初用于操作文件中的文本或从文件中提取数据和编写报告，但经过不断发展，现在可以操纵文件、进程并能够执行很多网络任务，已经逐渐成为了Web开发者用于处理表单的通用网关借口的实际脚本语言。

5.1.1 Perl文件处理

Perl脚本语言对于文本文件的处理功能十分强大，其中对文件的读写了解程度，是对Perl掌握程度的关键。接下来，我们将就Perl打开文件并进行相应的的读写、判断文件状态以及命令行参数和管道打开3部分展开详细阐述。

面试试题1

请谈谈你对Python脚本语言的文件处理功能的了解。

解答

Offer

Perl 脚本语言对于文件的处理功能可以大体分为以下 3 个部分。

- 文件的读写。
- 文件状态的判断。
- 命令行参数和管道。

要对文件进行读写，首先要先使用 open()函数打开文件，如果打开文件成功，则函数会返回非零结果；如果失败，则返回未定义的值。文件的读写均可通过该函数的不同参数进行控制。

说明

在 Win32 平台上打开文件，从磁盘读文本文件时，\r\n(回车和新行字符)转换为\n,而~Z 字符读作文件末尾标记符（EOF）。

面试试题 2

请解释 Perl 中的命令行参数并说明<>操作符的工作原理。

解答

像 C 一样，Perl 也有存储命令行参数的数组@ARGV，它可以用来分别处理各个命令行参数；与 C 不同的是，$ARGV[0]是第一个参数，而不是程序名本身。

```
$var = $ARGV[0]; # 第一个参数
$numargs = @ARGV; # 参数的个数
```

在 Perl 中，<>操作符实际上是对数组@ARGV 的隐含的引用，其工作原理介绍如下。

（1）当 Perl 解释器第一次看到<>时，打开以$ARGV[0]为文件名的文件。

（2）执行动作 Shift(@ARGV); 即把数组@ARGV 的元素向前移动一个，其元素数量即减少了一个。

（3）<>操作符读取在第一步打开的文件中的所有行。

（4）读完后，解释器回到第（1）步重复。

说明

为了能更好地理解<>操作符，我们可以分析一下实例。

```
@ARGV = ("myfile1", "myfile2"); #实际上由命令行参数赋值
while ($line = <>) {
    print ($line);
}
```

上述程序实际上是由命令行参数赋值，然后将文件 myfile1 和 myfile2 的内容打印到标准输出。



5.1.2 正则表达式

在学习了如何从文件中读取数据的方法后，再加上标量、数组和运算符方面的知识，就可以对数据进行操作了。但有时我们需要提取文件中有用的数据，这就需要借助 Perl 中的一个重要工具来执行这项任务，即正则表达式。

面试试题 1

请描述 Perl 语言中的正则表达式主要有哪几种存在形式。

解答

可以说正则表达式本身也可以称为是一种语言，它是用于描述要匹配模式的正式方法。在 Perl 程序中，正则表达式有 3 种存在形式，它们分别是以下几种。

- 匹配：m/<regexp>/ （还可以简写为 /<regexp>/ ，略去 m）
- 替换：s/<pattern>/<replacement>/
- 转化：tr/<pattern>/<replacemnt>/

上述 3 种形式一般都和 =~ 或 !~ 搭配使用（其中 “=~” 表示相匹配，在整条语句中读作 does；“!~” 表示不匹配，在整条语句中读作 doesn't），并在左侧有待处理的标量变量。如果没有该变量和 =~ !~ 操作符，则默认为处理 $_ 变量中的内容。

说明

在 Perl 的正则表达式中如果出现()，则发生匹配或替换后()内的模式被 Perl 解释器自动依次赋给系统 $1, $2

替换操作 s/<pattern>/<replacement>/ 还可以在末尾加上 e 或 g 参数，它们的含义分别为：s/<pattern>/<replacement>/g 表示把待处理字符串中所有符合 <pattern> 的模式全部替换为 <replacement> 字符串，而不是只替换第一个出现的模式。

为了更好地应用 Perl 的正则表达式，我们必须熟练掌握表 5-1 中的常用模式。

表 5-1 正则表达式中的常用模式

x?	匹配 0 次或一次 x 字符串
x*	匹配 0 次或多次 x 字符串，但匹配可能的最少次数
x+	匹配 1 次或多次 x 字符串，但匹配可能的最少次数
.*	匹配 1 次或多次的任何字符
{m}	匹配刚好是 m 个的指定字符串
{m,n}	匹配在 m 个以上 n 个以下的指定字符串
{m,}	匹配 m 个以上的指定字符串
[]	匹配符合 [] 内的字符

Offer

续表

[^]	匹配不符合 [] 内的字符
[0-9]	匹配所有数字字符
[a-z]	匹配所有小写字母字符
[^0-9]	匹配所有非数字字符
[^a-z]	匹配所有非小写字母字符
^	匹配字符开头的字符
$	匹配字符结尾的字符
\d	匹配一个数字的字符，和 [0-9] 语法一样
\d+	匹配多个数字字符串，和 [0-9]+ 语法一样
\D	非数字，其他同 \d
\D+	非数字，其他同 \d+
\w	英文字母或数字的字符串，和 [a-zA-Z0-9] 语法一样
\w+	和 [a-zA-Z0-9]+ 语法一样
\W	非英文字母或数字的字符串，和 [^a-zA-Z0-9] 语法一样
\W+	和 [^a-zA-Z0-9]+ 语法一样
\s	空格，和 [\n\t\r\f] 语法一样
\s+	和 [\n\t\r\f]+ 一样
\S	非空格，和 [^\n\t\r\f] 语法一样
\S+	和 [^\n\t\r\f]+ 语法一样
\b	匹配以英文字母、数字为边界的字符串
\B	匹配不以英文字母,数值为边界的字符串

面试试题 2

Perl 语言中的正则表达式对于模式匹配功能的原则有哪些？

解答

可以说正则表达式是 Perl 语言的灵魂，想学好 Perl 语言，就必须掌握正则表达式。虽然正则表达式是 Perl 的难点所在，不过一旦掌握其语法，就可以拥有几乎无限的模式匹配的能力，而且 Perl 编程的大部分工作都是围绕正则表达式进行的。

正则表达式在进行模式匹配时要遵循以下原则。

- 正则表达式有 3 种不同形式，即匹配、替换和转换。
- 正则表达式仅对标量进行匹配。
- 正则表达式匹配一个给定模式的最早的可能匹配。
- 正则表达式能够处理双引号所能处理的任意和全部字符。
- 正则表达式在求值过程中将会产生两种情况：结果状态和反向引用。

Offer

- 正则表达式的核心能力在于通配符和多重匹配运算符以及它们如何操作。
- 如果想匹配不止一个字符集合，则应在 Perl 中通过使用“|”来增加灵活性。
- Perl 用 (?..) 语法为正则表达式提供扩展功能。

强调

对于 Perl 语言的学习来说，能够熟练掌握文件的操作和正则表达式这两大部分就可以应用 Perl 进行开发工作了。Perl 的语法和数据类型很容易掌握。在实际的工作中要根据需求灵活运用以上知识，通过实践不断巩固提高。

5.2 Perl 测试起步

对于测试的作用我们都知道，它可以提高代码的可靠性和可维护性，甚至会优化软件开发的进程。读者可能也了解 Perl 所提供的可用模块的范围之大，不只是对于 Perl 程序的测试也包括非 Perl 程序的测试。简而言之就是，我们除了不知道如何开始进行 Perl 测试之外什么都知道。

通过本章的学习，读者可以掌握如何开始 Perl 测试、Perl 测试库是如何工作的，以及在哪里可以找到使工作容易的有用的库文件。在了解 Perl 测试如何开始之后，我们会继续了解更多的关于 Perl 测试方面的知识和技巧，最后我们可以写自己的 Perl 测试程序。在大家能够轻松自如地进行 Perl 的测试开发之前，我们举例说明一些基础知识。

5.2.1 安装及运行

面试试题 1

请谈谈对 Perl 模块安装的认识以及几种典型的模块安装方法。

解答

对于 Perl 测试程序的开发，我们应该先从测试模块的安装开始。CPAN（Comprehensive Perl Archive Network）是 Internet 上 Perl 模块最大的集散地，包含了至今公布的几乎所有的 Perl 模块。这些模块都是使用 Perl 解决的一些编程问题,它们以压缩文件的形式存在，为程序员更容易解决相似问题提供了代码库。不过在使用这些模块之前需要先安装，Perl 的模块安装操作十分简单，下面介绍常见的 3 种安装方法。

- 通过 CPAN shell 来安装

从 CPAN 安装模块的最好办法是使用包系统，它可以处理查找、下载和构建的所有细节问题，并且在安装模块的同时也安装该模块所依赖的模块。通常在类 UNIX 平台（包括 Mac OS X）以及带有 C 编译器的 Windows 平台会使用这种方法。下面以安装最新版本的 Test::Simple 模块

Offer

为例进行介绍。

安装前需要先登录 Internet，并且要取得 root 权限。perl-MCPAN-eshell 初次运行 CPAN 时需要做一些设置，如果您的机器是直接与 Internet 相连，则只需一直按回车键，并在最后选择一个离您最近的 CPAN 镜像站点即可。例如此时选择的是位于国内的中国自由软件库 ftp://freesoft.cgi.gov.cn/pub/languages/perl/CPAN。

通过以下几个步骤即可完成。

（1）获得帮助：cpan>h。

（2）列出 CPAN 上所有模块的列表：cpan>m。

（3）安装模块：cpan>install Test::Simple。

（4）退出：cpan>q。

• 通过 PPM 来安装。

在 Win32 下，建议使用 ActiveState 的 Perl for Win32，即 ActivePerl。ActivePerl 的主页为：http://www.activestate.com/。在 Win32 下，最好也使用专门为 ActivePerl 定制的 Perl 模块。从 CPAN 下载的 Perl 模块不能很好地在 ActivePerl 下使用。类似于 CPAN 模块，ActiveState 也开发了一个自动安装工具，称为 PPM（Perl Package Manager）。

最通常的安装方法是。

（1）首先联上线在 DOS 命令行下启动：ppm。

（2）获得帮助：PPM>h。

（3）列出 ActiveState 站点上所有为 Perl 模块的清单：PPM>search。

（4）安装模块：PPM>install Test::Simple。

（5）退出：PPM>q。

• 手动安装。

假如下载了 Test::Simple 模块 test-simple.tar.gz，并将其放在/usr/local/src/下。具体步骤如下。

（1）进入文件夹：cd /usr/local/src。

（2）解压缩这个文件：tar xvzf test-simple.tar.gz。

（3）这时会新建一个 test-simple 目录：cd test-simple。

（4）生成 makefile：perl Makefile.PL。

（5）建立模块：make。

（6）测试模块：make test。

强调

应根据所选用的平台采用相应的安装方法，具体情况具体对待，一定要灵活运用，全面掌握。对于 CPAN 站点主要有以下几个。

国内：ftp://freesoft.cgi.gov.cn/pub/languages/perl/CPAN 和 http://cpan.qz.fj.cn/

国外：http://www.cpan.org/ 和 http://www.perl.com/CPAN-local/



面试试题 2

对于 Perl 测试模块，如何全部和单独运行？请举例说明。

解答

以 Test::Harness 模块为例，假设我们已经从 CPAN 下载并解压到相应目录。

此时只需要使用 make test 命令即可运行所有从模块 Test::Harness 中找到的测试文件，并且给出总的运行结果。

```
$ make test
    PERL_DL_NONLAZY=1 /usr/bin/perl5.8.6 "-MExtUtils::Command::MM" "-e" \
        "test_harness(0, 'blib/lib', 'blib/arch')" t/*.t
    t/00compile.....ok 1/5# Testing Test::Harness 2.46
    t/00compile.....ok
    t/assert.....ok
    t/base.....ok
    t/callback.....ok
    t/harness.....ok
    t/inc_taint.....ok
    t/nonumbers.....ok
    t/ok.....ok
    t/pod.....ok
    t/prove-globbing....ok
    t/prove-switches....ok
    t/strap-analyze.....ok
    t/strap.....ok
    t/test-harness.....ok
            56/208 skipped: various reasons
    All tests successful, 56 subtests skipped.
    Files=14, Tests=551,  6 wallclock secs ( 4.52 cusr +  0.97 csys =  5.49 CPU)
```

如果想单独运行模块中的某一个测试文件，则可以使用 prove 命令。该命令只运行以命令行参数传入的文件，并且给出运行结果。例如下面所示的例子：

```
$ prove t/strap*.t
t/strap-analyze....ok
t/strap.....ok
All tests successful.
Files=2, Tests=284,  1 wallclock secs ( 0.66 cusr +  0.14 csys =  0.80
    CPU)
```

说明

当在一个文件中有很多测试需要做且有些运行失败时，为了更好地追踪错误，我们需要更详细的错误描述。此时我们需要在 prove 命令后面加上“详细”这个选项，即 prove –v。

面试试题 3

请分别解释 Test::More 模块中 use_ok()和 can_ok()的区别？

解答

首先我们要先了解一下 Test::More 这个模块，从该模块的名字看出，Test::More 可以做 Test::Simple 所能做的一切，并且能做更多的事情。其中较为关键的就是 use_ok()和 can_ok()这两个函数。

- use_ok()采用模块的名字来加载并且将一个可选列表的符号传递给模块的 import()方法。它会尝试加载模块、输入的符号并且基于结果给出所做测试是否成功。
- can_ok()用来检查模块中的函数是否存在于当前的名字空间。通常输入测试失败的原因有两点：要么输入失败，要么在测试文件中将名字打错。

强调

要进行 Perl 测试开发，必须掌握以下 3 个模块：Test::Simple、Test::More 和 Test::Harness。用户不仅要熟悉各个模块中所包括的函数，同时还要根据测试开发的需求灵活应用。

5.2.2 构建测试用例

面试试题 1

请使用 Test::Simple 模块中的 ok()函数写一个“Hello,world!”的测试程序。

解答

Test::Simple 是 Perl 测试开发中最常用的模块之一，其中的 ok()函数也是最简单的函数，所以我们选用这个例子作为我们的第一个 Perl 测试程序，希望读者能领会其中的用法。

```
#!perl

    use strict;
    use warnings;
    use Test::Simple tests => 1;

    sub hello_world
    {
        return "Hello, world!";
    }
    ok( hello_world(   ) eq "Hello, world!" );
保存这个程序，并使用 prove 运行
    $ prove hello.t
结果:
```

Offer!

```
hello....ok
All tests successful.
Files=1, Tests=1,  0 wallclock secs ( 0.09 cusr +  0.00 csys =  0.09 CPU)
```

上述程序中定义了一个简单的子程序 hello_world，这并没有用到特殊的语法。另一个子程序 ok()来自模块 Test::Simple，会报告运行测试的结果是成功还是失败，主要依据第一个参数的真假。在本例中，如果 hello_world 返回的字符串是“Hello, world!”则测试成功，否则失败。

5.3 Perl 测试管理

所有编程的一般规则都是适用于测试的：井井有条，尽量减少重复，更关注需求而不是技术实现的争论。对于小型项目，我们可以很容易地创建和管理一个测试文件，但对于大型或重要的项目，就需要花费更多的时间和精力对测试进行管理。把测试代码放置在哪里？如何组织测试文件？对于哪些选项你必须减少复杂性以及可控制的水平等？

本章将着重解释如何在测试套件中组织测试文件，了解和改善测试范围，编写自定义的测试库，并解释测试结果。

5.3.1 组织测试

编写测试文件很容易，但管理测试却不那么容易。如果运行完整的测试套件很难实现，那么即使有完整的测试代码覆盖率也没有任何用处。所以我们能进行以下操作。

- 测试代码自动化。
- 不需要用户的交互。
- 测试结果简洁明了。

这样就可以使测试代码不断重复运行，更有效地管理测试。

面试试题 1

请谈谈对 Perl 中组织测试的模块 Test::Harness 的认识以及如何按特定的顺序来运行测试？

解答

和 Perl 中其他的模块一样，Test::Harness 模块也要先从 CPAN 下载然后解压缩。运行 Makefile.PL 命令会为模块生成 Makefile，然后构建测试模块即可。

对于 Test::Harness 来说，首要条件是测试环境的搭建（如搭建虚拟的运行环境）和清理，选择运行单个测试或所有测试的方法，分析输出结果的手段及错误的标准报告。

此时需要注意模块中的 analyze_file()函数。测试中调用这个函数可以将所要测试的函数名作为参数，然后运行测试，最后收集解析测试结果并且返回关于该测试文件细节的哈希表，在该表中存放着关于测试结果的关键字，如表 5-2 所示。

Offer

表 5-2

关 键 字	描 述
Max	计划所运行的测试数目
Seen	实际执行的测试数目
Ok	通过的测试数目
Skip	跳过的测试数目
Todo	TODO 的测试数目
Bonus	通过的 TODO 测试数目

对于 ExtUtils::MakeMakeer 和 Module::Build 来说都是以特定的顺序来运行测试的。我们可以通过为测试文件的名字进行编号的方法来改变测试顺序。例如 00-file.t 将比 99-last.t 先被执行。另外还可以通过为测试文件建立子目录的方法来改变执行顺序。在默认的情况下测试文件是放在文件夹/t 下的。需要注意的是，应将所做的改变即创建了子目录通知处理器，这样测试文件就会按照目录逐层被执行了。

5.3.2 测试代码覆盖率

进行测试比没有测试好，但如果有充分的测试会更好。代码覆盖率是一种衡量代码有多少被实际被执行的一种很好的方法。手工分析代码覆盖率是“沉闷”的。幸运的是，我们可以从 CPAN 下载 Devel:: Cover 模块，它会自动分析代码覆盖率同时给出报告。更重要的是，它可以与标准的测试工具协同工作。

当模块 Devel::Cover 运行一个测试套件时，会配置 Perl 代码可选级别，使将要测试的代码中的每个字程序、语句、分支和条件都做检查，看是否都完全执行到。最后会将结果写到一个数据库文件中，在这个文件中，我们可以看到各种不同类型的覆盖率报告，这对于我们来说是非常重要的。

面试试题 1

对于 Perl 模块 Devel::Cover 来说，可以执行哪几种类型的代码覆盖率？请分别解释。

解答

Devel:: Cover 模块中主要有以下几种类型的覆盖。

- 语句覆盖：用来检查某个特定的语句是否被执行。例如语句$flag = 1 任何能够调用到该语句的测试就会在结果中被标记已经覆盖到了该语句。
- 分支覆盖：检查测试是否执行了分支语句的各个部分。例如代码 print “True!” if $flag;这个语句必须执行两次，即$flag 为真和$flag 为假两种情况，此时就可以认为覆盖率为 100%。
- 条件覆盖：需要考虑所有可能的逻辑条件。例如赋值语句：my $name = shift ||‘Ben’;测试必须传一个有实际值的字符串至少一次，传一次空字符串，还需要不传参数。当这 3 种情况都执行到了，条件覆盖率就为 100%。

Offer

- 子程序覆盖：检查测试至少执行了一个子程序。如果不能对某个特定模块执行完全测试，那么我们可以选择从某个子程序开始，只对该子程序生成报告即可。

说明

通常测试代码的覆盖率不可能都是100%的，有时我们要根据需要关注更为重要的代码覆盖率。在模块 Devel::Cover 中可以选择对哪种类型的覆盖生成报告。这样我们就可以有选择性地得到我们想要的报告了。注意：在模块 Devel::Cover 中使用 B::Deparse 对分支覆盖和条件覆盖生成输出报告。

5.3.3 编写并测试“测试库”

到现在为止，测试开发对我们来说已经不那么令人望而生畏了。它们只是程序，所有适用于一般程序的设计和组织在测试开发中同样适用。将所有相关的代码放在同一个地方，这对于 Perl 来说就是一个模块，即一个自成一体的代码库。

通过学习我们已经知道如何使用测试库了，这些测试库有些是 Perl 的标准库，有些则需要从 CPAN 上下载。如果我们发现自己解决的问题无法从现有的测试模块中找到答案，而这些问题又经常重复出现，就可以考虑编写自己的测试库了。模块 Test::Builder 使自己编写测试库成为可能。

Test::Builder 使得我们可以编写自己的测试库，同时对测试进行很好的管理。但归根结底自己编写的测试库也只是程序，好的测试库同样还需要好的测试。Test::Builder 也为测试编写的测试库做了很好的支持，通过使用 Test::Builder::Tester，我们就可以对测试库进行充分的测试以保证代码质量。

面试试题 1

请解释下面这段代码（该段代码为程序员自己编写的测试库），并写出一段简单程序来应用该测试库。

解答

```
package Test::Between;
use strict;
use warnings;
use base 'Exporter';
our@EXPORT = qw( is_between );
use Test::Builder;

    my $Test = Test::Builder->new(    );
    sub is_between ($$$;$)
    {
        my ($item, $lower, $upper, $desc) = @_;
        return
```

```
        (
            $Test->ok( "$lower" le "$item" && "$item" le "$upper", $desc ) || $Test->diag( "$item is not between
$lower and $upper" )
        );
    }
```

使用 Test::Builder 模块可以生成名为 Test::Between 的测试库。在这个库中使用了 Exporter 模块来输出函数 is_between()。这是一个简单的比较函数，它包括 3 个必需的参数：所要测试的值、测试值的上限、测试值的下限。另外还有一个可选的参数，其是对测试的说明。当所要测试的值大于或等于上限，或者小于或等于下限值，则函数返回真，否则为假。再通过模块 Test::Builder 提供的 ok()函数就可以对结果进行测试了。

我们现在已经了解这个测试库的基本功能了，那么写一个简单的测试程序来应用这个测试库就变得很容易了。在测试文件中使用 Test::Between 并在程序中调用 is_between()函数，然后输入参数就可以进行测试了。应用程序如下：

```
#!perl
use strict;
use warnings;
use Test::More tests => 3;
use Test::Between;
is_between('b','a','c','simple alpha compare');
is_between(2, 1 , 3 , 'simple numeric comparison');
is_between("two",1,3,'mixed comparison');package
```

在这个测试程序中主要进行了简单的字母比较和数字比较，同时还有混合比较。

前两个测试是成功的，第三个失败，运行的测试结果如下：

```
$ perl test_between.t
1..3
ok 1 - simple alphabetical comparison
ok 2 - simple numeric comparison

not ok 3 - mixed comparison
#       Failed test (examples/wtm_01.t at line 12)
#              two is not between 1 and 3
# Looks like you failed 1 test of 3.
```

强调

通常我们要把所编写的测试库放在指定的目录 lib/下，并把 Test::Between 保存为 Test/Between.pm，这样就可以在以后的程序中应用该测试库了。

Offer

面试试题 2

请使用 Test::Builder::Tester 对上道题目中的测试库进行测试，考虑测试的全面性，要求写出 3 种不同情况。

解答

通过上道题目我们已经知道该如何测试这个测试库了。这道题目考查的重点在于 Test::Builder::Tester 模块的应用，通过使用这个模块，我们可以在发布自己的测试库之前进行充分的测试。首先 Test::Builder::Tester 通过收集所要测试库的功能，然后运行测试并对实际结果和期望结果进行比较，最后打印出测试结果。这就要求我们对函数参数的功能必须十分了解，知道什么时候成功，什么时候失败。

和上例一样，我们设计 3 个测试，分别是对字母比较、数字比较和混合比较，前两个 is_between()调用是成功的，最后一个失败。

我们已经知道前两个 is_between()测试是成功的，那么此时就先调用 test_pass()函数来告知 Test::Builder::Tester 将会得到一个测试成功的结果描述，然后调用 is_between()进行比较并得到结果描述，最后调用 test_test()对两个结果进行比较即可得出结果是通过的。

第三个要测试 is_between()测试失败的情况，那么我们就要得到失败的结果描述，这是通过 test_fail()来得到的，然后进行测试，最后还是使用 test_test()来对结果进行比较，如果测试失败，那么返回的测试失败信息正是我们想要的，所以测试结果也是通过的。

具体的函数实现如下：

```
#!perl
use strict;
use warnings;
use Test::Between;
use Test::Builder::Tester tests => 3;

my $desc;

$desc = 'simple alphabetical comparison';
test_pass( $desc );
is_between( 'b', 'a', 'c',  $desc );
test_test( $desc );

$desc = 'simple numeric comparison';
test_pass( $desc );
is_between(  2,  1,  3, $desc );
test_test( $desc );

$desc = 'mixed comparison';
test_out( "not ok 1 - $desc" );
```

```
test_fail( +2 );
test_diag( '          two is not between 1 and 3' );
is_between(  "two",  1,  3, $desc                    );
test_test( 'failed comparison with diagnostics' );
```

面试试题3

调试测试库测试失败结果是很困难的，尤其是在很多测试用例的时候，请问有什么方法可以更直观、更准确地定位测试失败信息？

解答

在本题中我们可以使用 Test::Builder::Builder::Color，它继承自 Test::Builder::Tester，可以将输出的测试结果颜色化，使不同结果更容易被识别。因为它要求 Term::ANSIColor 模块的支持，所以也要安装这个模块。

有两种方法可以进行颜色调试，一种是在代码中使用该模块，即：

```
use Test::Builder::Tester::Color;
```

另一种是直接在命令行中使用如：

```
$ perl -MTest::Builder::Tester::Color between.t
```

强调

在本章中主要讲述了一些测试管理的基本内容，若想很好地掌握这些模块和方法，则还需要在实践中不断应用。总之测试的管理对于一个项目来说很重要，一定要提高重视程度。

5.4 Perl 单元测试

如果读者具有其他面向对象语言编程的经验，那么也许用过该语言的单元测试框架来开发测试用例和测试套件。面向对象单元测试框架对于像 C#和 Java 等面向对象语言来说是很通用的，而 Perl 语言的单元测试则主要是程序。这并不是说哪一种风格更优秀，而是说两者之间的选择要看具体的目标和软件框架。

支持 Perl 单元测试的模块主要有：Test::Unit、Test::SimpleUnit、Test::Class。每个模块都有自己的使用范围和优缺点，要根据实际需求来选择。

在本节中将主要介绍 Test:: Class 模块，它是一个非常强大的 Perl 测试库，其允许程序员设计 xUnit 类型的测试。使用 Test::Class 模块的测试是类，而不仅仅是测试文件，这使得单元测试的开始不那么容易，但是开发人员可以更好地组织测试用例，特别是可以尽量减少测试代码的重复性。

Offer

5.4.1 编写测试用例

Test::Class 模块为程序员提供一种简单的方法，即以 xUnit 风格创建类和对象来测试代码。通过使用 Test::Builder 构建测试程序可以使测试程序能与其他基于 Test::Build 的模块，如 Test::More,Test::differences, Test::Exception 等协同工作，甚至可以使单元测试实现自动化。

面试试题 1

请说出为什么要选择 Test::Class 模块进行单元测试？

解答

Test::Class 类可以提供简单的 xUnit 测试同时将 Perl 的标准测试风格集成进去，这是非常有用的。我们可以从以下几点更清楚地了解为什么要选择 Test:Class 模块进行单元测试。

- 该模块具有所有 xUnit 测试的优点。我们可以很容易地创建测试套件及相关测试。
- 该模块使用 Test::Builder 来构建并且可以和其他基于 Test::Builder 的模块很好地协同工作。这使得我们所写的测试类在*.t 脚本中同样可以运行，同时也使得将正常的测试重构为类更容易。
- 可以更容易地将测试代码打包为类或模块而不是*.t 脚本。
- 可以在测试类中有多个构造函数或析构函数。
- 可以使测试运行得更快。只要我们将*.t 脚本重构为测试类，就可以将所有的测试用例在一个脚本中运行，这样就可以省去每个脚本的启动时间。

当然，Test::Class 模块也不是万能的，在有些情况下就不适宜使用：

- 如果所有的*.t 脚本都运行良好，就不需要考虑使用 Test::Class，尤其对于不存在代码重用的简单的测试套件。
- 如果你所发布的代码恰巧是其他用户在其他模块中必须运行的，那么就不要选择使用 Test::Class。
- 如果习惯使用 TestCase/Suite/Runner 这些 JUnit 中常用的类结构，那么使用 Test::Unit 会更方便一些。

说明

关于 xUnit 测试框架。

xUnit 测试框架是继 SUnit 和 Junit（Sunit 是 Smaltalk 的测试框架，Junit 是 Java 的测试框架）之后适用于各种编程语言的通用测试框架。这些语言包括从 Ada 到 XSLT 的所有语言。xUnit 不只适用于单元测试，它对功能和验收测试也同样有用。

Test::Class 模块就是 Perl 中对 xUnit 测试框架的实现。

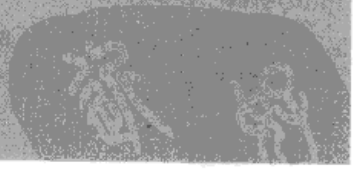
Offer

面试试题 2

请根据已知的 Queue::Test 测试类写一段测试程序，并写出运行结果。

```
package Queue::Test;

 use base 'Test::Class';
 use Queue;
 use Test::More;
 sub size : Test(4)
 {
       my $q1 = Queue->new(    );
       isa_ok( $q1, 'Queue' );
       is( $q1->size(    ), 0, 'an empty queue' );
       my $q2 = Queue->new(qw( howdy bonjour ));
       isa_ok( $q2, 'Queue' );
       is( $q2->size( ),2,'queue with some elements');
}
 sub enqueue : Test(2)
 {
       my $queue = Queue->new(    );
       isa_ok( $queue, 'Queue' );
       $queue->enqueue($_) for qw( howdy bonjour );
       is( $queue->size(    ), 2, 'queue is now larger' );
 }
 sub dequeue : Test(6)
 {
       my $queue = Queue->new(    );
       isa_ok( $queue, 'Queue' );
       is( $queue->dequeue, undef, 'empty queue' );
       $queue->enqueue($_) for qw( howdy bonjour );
       is( $queue->size(    ),2,'queue is now larger'    );
       is( $queue->dequeue(    ),'howdy','first    item');
       is( $queue->dequeue(    ),'bonjour','second item');
       is( $queue->size(    ), 0,'queue is now smaller' );
 }
```



解答

该段程序应用了模块 Test::Class、Queue 和 Test::More 实现了队列的测试类。对于队列的大小、出对和入队函数的测试分别给出了实现。在本节开始我们曾介绍过，Test::Class 可以是基于 Test::Builder 的任何测试模块，如 Test::Exception。大多数测试类都会使用 Test::More，本段程序中也有应用。在了解这段程序的功能后，我们就可以很容易地写出一段简单的测试代码，并将代码保存为 queue.t，具体实现可参看下面这段程序：

Offer

```
package Queue;
use Queue::Test;
use strict;
use warnings;
sub new
{
    my ($class, @items) = @_;
    bless \@items, $class;
}
sub size
{
    my ($self) = @_;
    return scalar @$self;
}
sub enqueue
{
    my ( $self, $item ) = @_;
    push @$self, $item;
}
sub dequeue
{
    my ( $self ) = @_;
    return shift @$self;
}
Test::Class->runtests(  );
```

这段测试程序 queue.t 所做的工作就是运行所有在 Queue::Test 中定义的方法。借助 Test::Class 可以跟踪所有 Queue::Test 的派生类。通过运行 Test::Class 中的 runtests()方法就可以完成对所有定义的方法的测试。

运行命令 prove queue.t，结果如下：

```
$ prove queue.t
    queue....#
    # Queue::Test->test_dequeue
    1..12
    ok 1 - The object isa Queue
    ok 2 - empty queue
    ok 3 - queue is now larger
    ok 4 - first item
    ok 5 - second item
    ok 6 - queue is now smaller
    #
    # Queue::Test->test_enqueue
    ok 7 - The object isa Queue
    ok 8 - queue is now larger
    #
```

```
# Queue::Test->test  size
ok 9 - The object isa Queue
ok 10 - an empty queue
ok 11 - The object isa Queue
ok 12 - a queue with some elements
ok
All tests successful.
Files=1, Tests=12,   1 wallclock secs ( 0.19 cusr +   0.00 csys =   0.19 CPU)
```

说明

我们需要注意测试类中的函数 enqueue : Test(n),Test(n)为该测试函数的属性，定义了该函数的类型，其中 n 定义了方法中要进行多少次测试。通常 Test::Class 模块会自动得出测试的总数，其中测试结果的 1..12 就表示要进行 12 个测试。

5.4.2 测试方法的类型

通过使用 Test::Class 可以定义不同类型的测试方法，其中主要包括以下 3 种类型。

- Test 方法。
- setup 和 teardown 方法。
- startup 和 shutdown 方法。

面试试题 3

请分别解释 Test::Class 中 3 种不同类型函数的用法。

解答

Test 方法：

一般是用来定义测试方法所用的属性。如下面这个函数：

```
sub subtraction : Test{
  is(2-1, 1, 'subtraction works');
};
```

定义了测试函数 subtraction 并且该测试运行一次测试。如果想在一个测试方法中运行多个测试，则只需要将测试数 n 写成 Test(n)即可。如果不知道要进行的测试数，则可以使用 Test(no_plan) 。

（1）setup 和 teardown 方法。

这两个方法是在每个测试前后运行的。如：

```
sub before : Test(setup)     { diag("running before test") };
sub after  : Test(teardown) { diag("running after test") };
```

我们可以使用这两个方法来创建一个在每个测试方法中都需要用的通用对象，即测试套件，并且可以将该对象存放在 Test::Class 中，使用时可以当做一个哈希表来处理。通过下例可以更

Offer

好地进行理解：

```
sub pig : Test(setup) {
    my $self = shift;
    $self->{test_pig} = Pig->new;
};

sub born_hungry : Test {
    my $pig = shift->{test_pig};
    is($pig->hungry, 'pigs are born hungry');
};

sub eats : Test(3) {
    my $pig = shift->{test_pig};
    ok(   $pig->feed,     'pig fed okay');
    ok(! $pig->hungry, 'fed pig not hungry');
    ok(! $pig->feed,     'cannot feed full pig');
};
```

（2）startup 和 shutdown 方法。

这两个方法与上两个方法有些相似，只不过这两个方法是对整个类来操作的。startup 方法在测试类开始前被执行，而 shutdown 方法则在整个测试类停止运行之前被执行。

这对于一些开销比较大的对象是很有用的，因为我们不想为每个单独测试都进行创建和销毁操作。下面这个数据库连接操作就是一个很好的例子：

```
sub db_connect : Test(startup) {
    shift->{dbi} = DBI->connect;
};

sub db_disconnect : Test(shutdown) {
    shift->{dbi}->disconnect;
};
```

5.4.3 SKIPPED 和 TODO 测试

在 Test::Class 中有两种测试非常有用，一种是 SKIPPED，即跳过测试；另一种是 TODO，即将测试标记为待测。这两种测试可以很好地帮助我们管理测试，SKIPPED 可以就整个类进行标记也可以对类中的某个函数单独操作，TODO 则只能对测试类中的某个函数进行标记而不能将整个类标记为待定，此时我们只需要使用 SKIPPED 即可达到这样的目的。

下面还以上面的 Queue::Test 为例，假如我们不想在下午 4 点以后运行该类中的所有测试，包括它的派生类，那么此时只需使用 Test::Class 中的 SKIP_CLASS()即可。下面来看具体代码实现：

```
package Queue::Test;
    use base 'Test::Class';
    use Queue;
```

```
use Test::More;

sub SKIP_CLASS
{
    return [ localtime(time) ]->[2] < 16 ? 0 : 'only runs      before tea time';
}

sub setup_queues : Test( setup => 2 )
{
   #  ...
}
```

运行结果如下（此时可以看到子函数 setup_queue()中的两个测试都被跳过）。

```
$ prove -v queue.t
queue....1..2
ok 1 # skip only runs before tea time
ok 2 # skip only runs before tea time
ok
        2/2 skipped: only runs before tea time
All tests successful, 2 subtests skipped.
Files=1, Tests=2,  0 wallclock secs (0.05 usr + 0.00 csys = 0.05 CPU)
```

面试试题 4

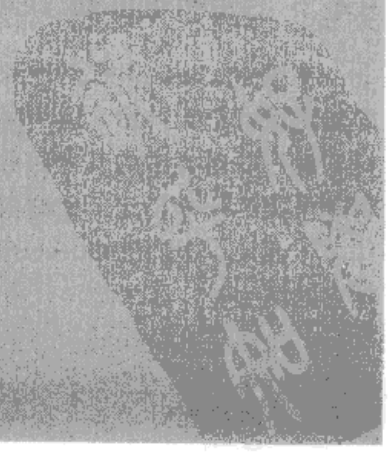
如果我们对特定类的测试跳过，前提是不覆盖 SKIP_CLASS()函数，能否实现？

解答

答案是肯定的。在不覆盖 SKIP_CLASS()函数的前提下，我们只需要将跳过测试的原因以参数的形式传给 SKIP_CLASS()函数即可。例如，我们不想让 Queue::Test 类中测试在上午进行，同时又不影响它的派生类，那么此时只需要在原来的测试类添加一个函数即可，具体实现如下：

```
package Queue::Test;
use base 'Test::Class';
use Queue;
use Test::More;
Queue::Test->SKIP_CLASS(
    [ localtime(time) ]->[2] <= 12
    ? 'only runs in the afternoon'
    : 0
);

sub size : Test(4)
{
   #  ...
}
```



说明

如果我们想对某个测试类中的特定测试跳过，则可以在剩余测试都运行完之前在方法中返回。而返回值就是为什么跳过该测试的原因。下面两句代码可以很清楚地说明这个问题，还是以队列为例。

```
can_ok($queue-enqueue, 'enqueue') or return("enqueue failed");
.....
```

如果测试类中存在函数 enqueue()，但运行时失败了，则 can_ok()函数就会返回 "enuqeue failed" ,同时紧接着的测试都会跳过。

5.5 Perl 数据库测试

很多的应用程序都需要和外部数据进行交互，要么使用数据库，要么使用简单的文件，又或是介于两者之间。现在的应用程序在出于对数据的存储安全性、抽象性及可维护性等方面的考虑，通常会选择使用数据库。数据库的使用可以使数据存储和应用程序得到一定程度的分离，使应用层不用考虑太多下层的存贮。这本身是非常好的解决方案，但同时也对测试提出了更大的挑战。因为在应用程序之外的东西一般来说是很难被测试的。如何知道是否连接数据库成功？如何确定将使用数据库的哪个用户去登录？

幸运的是在 Perl 的 DBI 模块中，一些测试工具使得对数据库的测试变得简单，我们可以确定不论是数据库内部还是外部的操作都具有正确性。本章中我们将学会如何装载并测试数据库以及模拟数据库等多种测试数据库的方法，这对提高我们的测试水平是很有帮助的。

5.5.1 装载并测试数据库

通常我们只需对于一个具有测试数据的简单数据库进行相关测试即可。DBI 模块和几个简单易用的数据库驱动模块一起工作，这些驱动包括 DBD::CSV 和 DBD::AnyData。这些驱动和 DBI 可以为我们已有的关系数据库提供统一接口用于测试。如果已经在同一个程序中对数据库的创建和连接进行了抽象化，那么我们就可以控制或模拟数据库的操作了，也可以创建一个数据库句柄用于测试，这可以实现使用代码代替实际的数据库连接。

面试试题 1

先读懂下段代码，然后应用 DBI 和相关驱动写一段简单的测试代码。

解答

```
package Users;
use strict;
```



```
use warnings;
my $dbh;

    sub set_db
    {
        my ($self, $connection) = @_;
        $dbh                    = $connection;
    }

    sub fetch
    {
        my ($self, $column, $value) = @_;
        my $sth = $dbh->prepare(
            "SELECT id, name, age FROM users WHERE $column = ?" );
        $sth->execute( $value );
        return unless my ($id, $name, $age) = $sth->fetchrow_array(  );
        bless { id => $id, name => $name, age => $age, _db => $self }, 'User';
    }

    sub create
    {
        my ($self, %attributes) = @_;
        my $sth                 = $dbh->prepare(
            'INSERT INTO users (name, age) VALUES (?, ?)'
        );
        $sth->execute( @attributes{qw( name age )} );
        $attributes{id} = $dbh->last_insert_id( undef, undef, 'users', 'id' );
        bless \%attributes, 'User';

    }
    package User;
    our $AUTOLOAD;
    sub AUTOLOAD
    {
        my $self    = shift;
        my ($member) = $AUTOLOAD =~ /::(\w+)\z/;
        return $self->{$member} if exists $self->{$member};
    }
```

首先，我们来分析一下这段代码。这个 Users 模块主要的功能就是从一个单表中创建并获取用户信息，包括姓名、年龄和用户 ID。我们可以把这段代码看成是一些数据库操作的包装层。需要注意的是函数 set_db()，它为整个类存储了一个数据库句柄。在了解代码功能后，写一段测试代码就很简单了。测试代码如下：

```
#!perl
use strict;
```

Offer

```
use warnings;
use DBI;

my $dbh = DBI->connect( 'dbi:SQLite:dbname=test_data' );
{
    local $/ = ";\n";
    $dbh->do( $_ ) while <DATA>;
}//应用 DBI 创建连接

use Test::More tests => 10;

    my $module = 'Users';
    use_ok( $module ) or exit;
    can_ok( $module, 'set_db' );

$module->set_db( $dbh );
//测试 fetch 用户
can_ok( $module, 'fetch'   );
    my $user = $module->fetch( id => 1 );
    isa_ok( $user, 'User' );
    is( $user->name(   ), 'Randy', 'fetch(   ) should fetch proper user by id' );
    $user      = $module->fetch( name => 'Ben' );
is( $user->id(   ), 2, '... or by name' );
//测试创建用户
    can_ok( $module, 'create' );
    $user      = $module->create( name => 'Emily', age => 23 );
    isa_ok( $user, 'User' );

    is( $user->name(   ), 'Emily', 'create(   ) should create and return new User' );
is( $user->id(   ), 3, '... with the correct id' );

    __END__
//创建用户表
    BEGIN TRANSACTION;
    DROP TABLE users;
    CREATE TABLE users (
        id    int,
        name varchar(25),
        age   int
    );

    INSERT INTO "users" VALUES(1, 'Randy', 27);
    INSERT INTO "users" VALUES(2, 'Ben', 29);
    COMMIT;
```

这段测试代码先装载了 DBI 模块并应用驱动 DBD::SQLite 创建了一个 SQLite 数据库连接。

然后执行文件末尾的 SQL 命令，创建了一个 users 表，并在表中插入了两条测试数据。

测试再调用 Users->set_db()创建了$dbh 句柄，以后所有的队 Users 类中函数的调用都要使用这个句柄。紧接着就对 fetch()和 create()方法的测试并检查它们的返回值。

这个题目考查的关键点在于对 DBI 和数据库驱动的理解，这种测试方法值得借鉴。

说明

SQLite 是一种简单但功能强大的关系数据库，它将所有的数据都存储在单一的文件中。

面试试题 2

我们知道只有一些最基本的简单 SQL 查询是可以在不同的数据库中兼容的。如果代码中使用不可移植或特定数据库功能，那么可否还使用上例中的方法即 DBI 和相关驱动来进行测试？

解答

上例中所用到的技术主要使用 SQL 及其子集对一些可以在主流数据库中可移植的操作进行测试。如果应用程序使用一些 SQL 超集、特殊数据库架构或存储过程，那么只使用 DBD::SQLite 或 DBD::AnyData 是不能够进行测试的。在这种情况下有两种方法可以解决这个问题：即要么使用一个具有完整测试数据的同等数据库来进行测试，要么采用模拟数据库的方法。

5.5.2 使用临时数据库

有些应用程序会依赖特定数据库的功能。例如，对数据库 PostgreSQL 或 MySQL 的管理应用就需要对数据库有很深入的了解。对于其他的一些程序，如网页内容管理系统，就需要创建自己的表并需要进行插入配置数据到数据库。在测试这样的数据库只使用 DBD::CSV 是不够的，因为无法覆盖整个系统。

对于这种情况，最好的办法就是针对实际的数据库进行测试，至少也要有一个具有实际数据的数据库。如果已经创建了数据库表，那么只要再进一步用同样的信息创建一个测试数据库，那么我们就可以应用这个测试数据库进行测试了。

面试试题 3

假如已有应用程序 My::App（保存在 lib/My/App.pm）和文件 sql/schema.sql（在文件中保存了数据库架构和一些基本数据），如果想在安装模块的过程中同时创建应用数据库和测试数据库，并且要创建连接，有什么办法可以实现？

解答

当应用程序需要和特定数据库进行交互时，想要对程序进行充分的测试，解决这个问题是

Offer

不可避免的。通常对于此类问题，我们会采用创建临时数据库的方法。一般来说我们需要创建 Module::Build 子类来对数据库用户信息的配置和数据库的安装。可以把文件保存成 build_lib/MyBuild.pm。在该文件中实现配置文件 create_config_file（），可对数据库用户名、密码、实际数据库名称以及测试数据库名称等信息进行配置。另外还有两个重要的子函数 ACTION_build（）和 ACTION_test（），这两个函数主要是根据配置文件创建不同的数据库。在应用 lib 'build_lib'和 MyBuild 模块写出自己的构建脚本 Build.PL。至于测试文件只需使用配置文件中的信息创建测试数据库，再使用 DBI 测试连接即可。请看以下程序：

```
package MyBuild;
 use base 'Module::Build';
 use DBI;
 use File::Path;
 use Data::Dumper;
 use File::Spec::Functions;
 //创建配置文件
 sub create_config_file
 {
     my $self       = shift;
     my $config     =
     {
         db_type   => $self->prompt( 'Database type ',          'SQLite'     ),
         user      => $self->prompt( 'Database user: ',         'root'       ),
         password => $self->prompt( 'Database password: ',   's3kr1+'     ),
         db_name   => $self->prompt( 'Database name: ',         'app_data' ),
         test_db   => $self->prompt( 'Test database name: ', 'test_db'    ),

     };
     $self->notes( db_config      => $config );
     mkpath( catdir( qw( lib My App ) ) );
     my $dd           = Data::Dumper->new( [ $config ], [ 'db_config' ] );

     my $path       = catfile(qw( lib My App Config.pm ));
     open( my $file, '>', $path ) or die "Cannot write to '$path': $!\n";
     printf $file <<'END_HERE', $dd->Dump(    );

 package My::App::Config;
 my $db_config;
%s
 //生成配置文件
 sub config
 {
     my ($self, $key) = @_;
     return $db_config->{$key} if exists $db_config->{$key};
 }
```

```
END_HERE
}
 //应用配置文件创建数据库
sub create_database

{
    my ($self, $dbname) = @_;
    my $config              = $self->notes( 'db_config' );
    my $dbpath              = catfile( 't', $dbname );
    local $/                = ";\n";
    local @ARGV             = catfile(qw( sql schema.sql ));
    my @sql                 = <>;
    //创建数据库连接句柄
    my $dbh                 = DBI->connect(
        "DBI:$config->{db_type}:dbname=$dbpath",
        @$config{qw( user password )}
    );
    $dbh->do( $_ ) for @sql;
  }
//创建实际数据库
sub ACTION_build
{
    my $self   = shift;
    my $config = $self->notes( 'db_config' );
    $self->create_database( $config->{db_name} );
    $self->SUPER::ACTION_build( @_ );
}
//创建测试数据库
sub ACTION_test
{
    my $self   = shift;
    my $config = $self->notes( 'db_config' );
    $self->create_database( $config->{test_db} );
    $self->SUPER::ACTION_test( @_ );
}
```

下面文件为 Build.PL，保存后运行命令 perl Build.PL，再运行命令 perl Build 会创建数据库‘app_data’以及 Config.pm 和 App.pm 两个执行脚本：

```
 #!perl
use strict;
use warnings;
use lib 'build_lib';
use MyBuild;
```



```
my $build = MyBuild->new(
    module_name      => 'My::App',
    requires         =>
    {
        'DBI'              => '',
        'DBD::SQLite' => '',
    },
    build_requires =>
    {
        'Test::Simple' => '',
    },
);
$build->create_config_file(  );
$build->create_build_script(  );
```

下段为测试程序 t/myapp.t，在该程序中只有一个测试用例，应用配置脚本创建数据库并测试连接：

```
#!perl
//更改执行目录到/t
BEGIN
{
    chdir 't' if -d 't';
}
use strict;
use warnings;
use Test::More 'no_plan'; # tests => 1;
use DBI;
use My::App::Config;
//配置数据库
my $user      = My::App::Config->config( 'user'        );
my $pass      = My::App::Config->config( 'password' );
my $db_name = My::App::Config->config( 'test_db'   );
my $db_type = My::App::Config->config( 'db_type'   );
//连接数据库，创建数据库句柄
my $dbh       = DBI->connect( "DBI:$db_type:dbname=$db_name", $user, $pass );

my $module   = 'My::App';
use_ok( $module ) or exit;
```

如果用户运行 Perl Build 程序将会创建实际应用数据库；如果运行 Perl Build test 将只会创建测试数据库。

强调

对于本题中所介绍到的创建临时数据库的办法，需要读者在实际中多加应用，关键是对

整个测试框架的掌握。

5.5.3 模拟数据库

严谨的程序如果和外部程序交互都需要对错误进行处理。对于数据库程序来说，这就更为重要了。当数据库失去连接会发生什么?当程序崩溃时丢失数据，该如何处理？

错误处理既然如此重要，那么对错误处理的测试就更为重要了。想要模拟数据库故障并不容易，幸运的是 Perl 在模块 DBD::Mock 中提供了模拟数据库的方法，这样以后再解决这样的问题就简单得多了。

面试试题 4

已知 Perl 简单模块 InsertWrapper，该模块主要是登录数据库和向数据库中插入数据两个功能。请思考如何解决测试程序不能连接数据库或连接异常的情况?

解答

```
package InsertWrapper;
use strict;
use warnings;
use DBI;
//创建数据库连接
sub new
{
    my ($class, %args) = @_;
    my $dbh              = DBI->connect(
        @args{qw( dsn user password )},
        { RaiseError => 1, PrintError => 0 }
    );
    my $self = bless { dbh => $dbh, logfh => $args{logfh} }, $class;
    $self->log( 'CONNECT', dsn => $args{dsn} );
    return $self;
}
//获取数据库句柄
sub dbh
{
    my $self = shift;
    return $self->{dbh};
}
//打印日志文件
sub log
{
    my ($self, $type, %args) = @_;
    my $logfh                = $self->{logfh};
```

Offer!

```
        printf {$logfh} "[%s] %s\n", scalar( localtime(  ) ), $type;
        while (my ($column, $value) = each %args)
        {
            printf {$logfh} "\t%s => %s\n", $column, $value;
        }
}
    //插入数据
    sub insert
    {
        my ($self, $table, %args) = @_;
        my $dbh                    = $self->dbh(  );
        my $columns                = join(', ', keys %args);
        my $placeholders           = join(', ', ('?') x values %args);
        my $sth                    = $dbh->prepare(
            "INSERT INTO $table ($columns) VALUES ($placeholders)"
        );
        $sth->execute( values %args );
        $self->log( INSERT => %args );
    }
```

对于该模块的测试应该主要集中在：connect()和 insert()这两个函数是否可以正常工作，此时需要考虑数据库正常工作和异常两种情况。对于这个模块的测试可以采用模拟数据库的方法，这对于数据库连接失败或异常会非常有用。在测试文件中我们会用到 DBD::Mock 模拟数据库，Test::Exception 处理异常，以及 Test::More 进行测试用例开发。具体实现如下：

```
#!perl
use strict;
use warnings;
use IO::Scalar;
use Test::More tests => 15;
use DBD::Mock;
use Test::Exception;
//测试模块是否存在
my $module      = 'InsertWrapper';
use_ok( $module ) or exit;

my $log_message = '';
my $fh          = IO::Scalar->new( \$log_message );
my $drh         = DBI->install_driver( 'Mock' );
can_ok( $module, 'new' );

//测试模拟连接是否成功
$drh->{mock_connect_fail} = 1;

my %args = ( dsn => 'dbi:Mock:', logfh => $fh, user => '', password => '' );
throws_ok { $module->new( %args ) } qr/Could not connect/,
```

```
        'new(    ) should fail if DB connection fails';
$drh->{mock_connect_fail} = 0;

    my $wrap;
lives_ok { $wrap = $module->new( %args ) }
    '... or should succeed if connection works';

isa_ok( $wrap, $module );

like( $log_message, qr/CONNECT/,                '... logging connect message' );
like( $log_message, qr/\tdsn => $args{dsn}/, '... with dsn'                    );
$log_message = '';

can_ok( $module, 'dbh' );
isa_ok( $wrap->dbh(   ), 'DBI::db' );
//测试向模拟数据库插入数据是否成功
can_ok( $module, 'insert' );
$wrap->dbh(   )->{mock_can_connect} = 0;


throws_ok { $wrap->insert( 'users', name => 'Jerry', age => 44 ) }
    qr/prepare failed/,
    'insert(    ) should throw exception if prepare fails';

$wrap->dbh(   )->{mock_can_connect} = 1;

lives_ok { $wrap->insert( 'users', name => 'Jerry', age => 44 ) }
    '... but should continue if it succeeds';

like( $log_message, qr/INSERT/,                 '... logging insert message' );
like( $log_message, qr/\tname => Jerry/, '... with inserted data'       );
like( $log_message, qr/\tage => 44/,        '... for each column'           );
```

5.6 Perl 数据驱动测试框架

对于 Perl 模块开发来说最困难的部分就是对模块的测试了。因为模块是提供给其他程序使用的，所以对于测试的要求就更高。对此 Perl 提供了标准的运行测试模块 Teat::Haress 和基本的测试用例开发模块 Test::More。在使用这些模块的过程中我们会发现这远不能满足测试的要求，所以我们会想去开发自己的测试框架。虽然使用 Test::More 可以通过实现 Test::Builder 子类来创建定制的测试框架，但这并不容易实现。Perl 提供了一种更简单的方法以便于开发人员创建自己的测试框架，简单到只需要在所写的框架中使用模块 Test::Base 即可。



5.6.1 Test::Base 框架

Test::Base 是一种基于数据驱动的测试框架。Test::Base 的实质就是通过过滤器来生成数据、执行被测试函数，最后与预期结果比较。

面试试题 1

请分析下面应用 Test::Base 框架所开发的测试程序。

Test.pm 是应用 Test::Base 的一个模块，也是我们需要测试的模块：

```
# lib/MyProject/Test.pm
package MyProject::Test;
use Test::Base -Base;

use MyProject;

package MyProject::Test::Filter;
use Test::Base::Filter -base;
 //实现自己的过滤器
sub my_filter {
    return MyProject->do_something(shift);
}
```

测试程序如下，保存为 sample.t:

```
# t/sample.t
use MyProject::Test;

plan tests => 1 * blocks;

run_is input => 'expected';

sub local_filter {
    s/my/your/;
}

__END__
//测试块一
=== Test one (the name of the test)
--- input my_filter local_filter
my
input
lines
--- expected
expected
```



```
output
//测试块二
=== Test two
This is an optional description
of this particular test.
--- input my_filter
other
input
lines
--- expected
other expected
 output
```

解答

在测试框架 Test.pm 中我们引用了 Test::Base，随后定义了一个过滤器。而在测试程序中我们引用了模块 MyProject::Test，此时就可以使用 Test::Base 对框架进行测试。__END__标记之后是测试数据，测试数据格式如下：

```
=== 测试数据标题
--- <输入数据名> [<过滤器 1> <过滤器 2> ...]
输入数据
--- <预期输出数据名> [<过滤器 1> <过滤器 2> ...]
预期输出数据
```

由于测试数据标题只是用来标识一个数据的开始，所以标题可以随便写。数据名就是 run_is 函数中使用的名称，在本例中，'input'为输入，'expected'为预期输出。

接着要说明的就是过滤器。Test::Base 接收的测试数据都是成对的——一个是输入，另一个是预期的输出，而 Test::Base 会使用输入数据运行被测试函数，如果得到的输出结果等于预期输出，则判定为 ok，否则为 fail。

在默认情况下，传递给 Test::Base 的测试数据都是字符串，显然不能满足各种各样测试的要求，所以 Test::Base 提供了 filter 这个概念，可以通过 filter 将字符串类型的测试数据转换成所需类型，然后再进行测试。再进一步，可以将被测试函数也做成一个 filter，这样输入数据和预期输出数据分别通过各自的过滤器后得到的结果应该相等。

面试试题 2

请比较 Test::More 和 Test::Base 模块，如果要进行自动化测试开发，选择哪个模块更好些？

解答

首先，Test::Base 和 Test::More 完全兼容，也就是说，用户可以只将 use Test::More;换成 use Test::Bas 即可，而不用做任何代码改动。

其次，Test::Base 扩展了 Test::More，在可以使用所有 Test::More 的函数之外，还增加了很

Offer

多自己的函数。如果熟悉 Test::More 的测试开发，那么还是可以使用原来的一些方法进行测试开发的，这样就为开发人员提供了很大的便利。

最后，Test::Base 可以提供更简单的、可重用的数据驱动测试方法，让程序员不必在烦琐的测试程序上花费太多时间。程序员只需应用 Test::Base 框架，写好输入和期望输出结果即可。注意，此时对于输入和输出结果也有一定的规范。

综上所述，在进行自动化测试开发过程中，Test::Base 是更好的选择。

5.6.2 Test::Base 主要函数

在 Test::Base 模块中有很多有用的函数，如果想更好地应用这个框架进行自动化测试开发，用户必须很好地掌握其中的函数。Test::More 中的函数在 Test::Base 都可以使用，有些函数在 Test::Base 中还做了扩展。以 is(actual, expected, [test-name])为例，它和 Test::More 模块中的 is 函数是等同的，不过增加了一个新的功能，当实际结果和期望结果不同并且输出为多行时，这个函数会以统一的格式来输出差异。当我们需要在几百行不同的输出中寻找一个字母时，统一的格式会为我们带来很多的好处。差异输出需要 Text::Diff 模块的支持，如果没有安装该模块，那么 is 函数就会和 Test::More 中的输出相同。

Test::Base 模块中有以下几个函数需要掌握，具体用法可参照 CPAN 网站。

- is(actual, expected, [test-name])。
- blocks([data-section-name])。
- next_block()。
- first_block()。
- run(&subroutine)。
- run_is([data_name1, data_name2])。
- is_deep($data1, $data2, $test_name)。
- run_is_deeply([data_name1, data_name2])。
- run_is_deeply([data_name1, data_name2])。
- run_is_deeply([data_name1, data_name2])。
- run_like([data_name, regexp | data_name])。
- run_unlike([data_name, regexp | data_name])。
- run_compare(data_name1, data_name2)。
- delimiters($block_delimiter, $data_delimiter)。
- spec_file($file_name)。
- spec_string($test_data)。
- filters(@filters_list or $filters_hashref)。
- filters_delay([1 | 0])。
- filter_arguments()。

Offer

- tie_output()。
- no_diff()。
- default_object()。
- WWW() XXX() YYY() ZZZ()。
- croak() carp() cluck() confess()。

面试试题3

请解释 Test::Base 模块中 blocks([data-section-name])函数的用法。

解答

Test::Base 模块中最重要的函数就是 blocks。在列表上下文情况下，函数会返回 Test::Base::Block 列表，该列表是根据测试文件的数据段的测试规范生成的。在标量的上下文情况下，函数会返回对象的个数，这对于计算测试块的个数是很有用的。

blocks 函数可以带一个可选的参数，该参数表示仅返回包含特定名称的数据段。在默认情况下函数会返回所有的块。下面列出参数的两种不同用法：

```
my @all_of_my_blocks = blocks;
my @just_the_foo_blocks = blocks('foo');
```

说明

对 Test::Base 函数的掌握程度可以考察面试者对框架的理解，所以用户应多应用多总结，这样才能对函数有更深的了解。

5.6.3 Test::Base 测试规范

Test::Base 允许开发人员在外部文件中定义测试数据，程序的数据段或变量会包含所有的文本输入。可以说，测试规范就是一些文本行。每一个测试用例（或测试块）都由一行包含块分隔符和可选的测试名称分离开来。每一个块在进一步被分成命名段，这是由一行包含数据分隔符和数据段名称的行来实现的。这样的好处是，我们不用花费太多精力去研究测试程序，而是专注于测试数据的全面性。

下面是基本的测试规范：

```
=== <block name 1>//测试块名字
<optional block description lines>
--- <data section name 1> <filter-1> <filter-2> <filter-n>//输入数据段名称和过滤器
<test data lines>//实际的输入值
--- <data section name 2> <filter-1> <filter-2> <filter-n>//输出数据段名称和过滤器
<test data lines>//实际输出值
--- <data section name n> <filter-1> <filter-2> <filter-n>
<test data lines>
```

Offer

```
=== <block name 2>
<optional block description lines>
--- <data section name 1> <filter-1> <filter-2> <filter-n>
<test data lines>
--- <data section name 2> <filter-1> <filter-2> <filter-n>
<test data lines>
--- <data section name n> <filter-1> <filter-2> <filter-n>
<test data lines>
```

默认的数据块分隔符为===，默认的数据分隔符为---。

面试试题 4

请解释用于程序控制的特殊的数据段名字：SKIP、ONLY、LAST 的用途。

解答

- SKIP: 标记为 SKIP 的测试块会将该块中所有的测试忽略。这主要用来暂时关闭测试。
- ONLY: 标记为 ONLY 的测试块会使程序只执行该块的测试。当我们只想测试某一块的测试是否通过时，只需使用 ONLY 即可。这对于调试测试代码十分有用。
- LAST: 标记为 LAST 的测试块会使得程序只运行测试规范中最后一个测试块，而其他的测试块都会被忽略。

5.6.4 Test::Base 过滤器

使用 Test::Base 写测试用例真正的强大之处就在于该模块的过滤能力。Test::Base 具有一个不断增长的、有用的通用适配器，可以设置适配器顺序并且可以将其应用到不同的测试模块中。这意味着我们可以用最具可读性的格式来指定测试块顺序，然后通过使用过滤器把它翻译成测试中实际需要的数据。写自己的过滤器也是很简单的。

Test::Base 支持为每一个测试块的每一个数据段指定过滤器列表。默认的过滤器为 norm 和 trim。这些过滤器将按顺序应用到数据中，即在过滤器按照测试规范被解析之后，在被设置到 Test::Base 对象之前。我们可以使用 filters 函数将默认的过滤器列表添加到程序中，同时还可以通过在数据段分隔行中数据段名称后列出过滤器名称以制定附加的过滤器。如下例所示：

```
use Test::Base;
filters qw(foo bar);//使用函数添加过滤器列表
filters { perl => 'strict' };

sub upper { uc(shift) }//自定义的过滤器
__END__//标示以下为数据段

=== Test one
--- foo trim chomp upper//upper 为自订制的过滤器
```

Offer

...

面试试题5

请解释如下过滤器列表：norm、trim、lines、chomp、array、dumper、eval。

解答

数据段总是以单独的字符串标量开始的。norm 需要一个标量并且返回一个标量；trim 需要一个列表并且返回一个列表，但标量也是一个有效的列表；lines 需要一个标量并返回一个列表；chomp 需要一个列表并返回一个列表；array 需要一个列表并返回一个标量（一个匿名数组引用包含列表元素）；dump 需要一个列表并返回一个标量；eval 需要一个标量并返回一个列表。

强调

只有一个元素的列表可以作为需要标量的过滤器的输入，但其他的列表如果作为过滤器的输入就会出现异常。一个标量在列表的上下文可以被看作是只有一个元素的列表，也可以正常工作。

Offer

第6章 软件测试面试 Python语言考查

6.1 Python 语言考查

Python，是一种面向对象的解释性的计算机程序设计语言，也是一种功能强大而完善的通用型语言，已经具有十多年的发展历史，成熟且稳定。Python 具有脚本语言中最丰富和强大的类库，足以支持绝大多数日常应用。这种语言具有非常简捷而清晰的语法特点，适合完成各种高层任务，几乎可以在所有的操作系统中运行。目前，基于这种语言的相关技术正在飞速的发展，用户数量急剧扩大，相关的资源也非常多。

Python 语言的创始人是 Guido van Rossum。之所以选中 Python（大蟒蛇的意思）作为程序的名字，是因为他是一个 Monty Python 的飞行马戏团的爱好者。也许他也没有想到 Python 会在工业和科研上获得如此广泛的使用。对于那些从来没有学习过编程或并非计算机专业的编程学习者而言，Python 是最好的选择之一。

6.1.1 Python 总体考查

Python 是一门跨平台的脚本语言，它规定了一个 Python 语法规则，实现了 Python 语法的解释程序就成为了 Python 的解释器。目前我们用的比较多的是 C 版本的 Python，也就是使用 C 语言实现的 Python 解释器，除此之外还有使用 Java 实现的 Jython 和使用.NET 实现的 IronPython，这些实现可以使 Python 用户充分利用已有的 Java 及.NET 资源。可扩充性可以说是 Python 作为一种编程语言的特色。新的内置模块（module）可以用 C 或 C++写成，而我们也可为现成的模块加上 Python 的界面。Python 可以使用户避免过分的语法的羁绊，而将精力主要集中到所要实现的程序任务上。因此 Python 具有非常庞大的程序库，通过它们可以快速完成绝大部分常用的任务。

面试试题 1

请谈谈 Python 脚本语言的优缺点。



解答

Python 脚本语言的优点如下。

- 简单：Python 是一种代表简单主义思想的语言。阅读一个良好的 Python 程序就感觉像是在读英语一样，Python 的这种伪代码本质是它最大的优点之一。
- 易学：就如同你即将看到的一样，Python 极其容易上手。
- 免费、开源：Python 是 FLOSS（自由/开放源码软件）之一。简单地说，你可以自由地发布这个软件的拷贝、阅读它的源代码、对它做改动、把它的一部分用于新的自由软件中。
- 高层语言：当你用 Python 语言编写程序时，你无须考虑诸如如何管理你的程序使用的内存一类的底层细节。
- 可移植性：由于它的开源本质，Python 已经被移植在许多平台上，可以说 Python 几乎适应各种平台。
- 解释性：Python 语言写的程序不需要编译成二进制代码，可以直接从源代码运行。在计算机内部，Python 解释器把源代码转换成称为字节码的中间形式，然后再把它翻译成计算机使用的机器语言并运行。
- 面向对象：Python 既支持面向过程的编程也支持面向对象的编程。与其他主要的语言如 C++和 Java 相比，Python 以一种非常强大且又简单的方式实现面向对象编程。
- 可扩展性：如果你希望某段关键代码运行得更快或者希望某些算法不公开，则可以将部分程序用 C 或 C++编写，然后在 Python 程序中使用它们。
- 可嵌入性：你可以把 Python 嵌入你的 C/C++程序，从而向你的程序用户提供脚本功能。
- 丰富的库：Python 标准库十分庞大。它可以帮助你处理各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV 文件、密码系统、GUI（图形用户界面）、Tk 和其他与系统有关的操作。

Python 脚本语言的缺点如下：

任何编程都不是完美的，都有局限性，Python 也不例外。

- 运行速度慢：Python 虽然比 Java 运行速度快，但仍无法和需要编译的语言相比。如果有速度要求，则最好用 C++改写关键部分。
- 国内市场较小：一直以来国内以 Python 来做主要开发的只有一些 Web2.0 公司。但随着时间的推移，目前很多国内软件公司，尤其是游戏公司，也开始有规模地使用该语言。
- 中文资料匮乏：好的 Python 中文资料屈指可数。为了了解最新的 Python 语言发展，应最好阅读英文资料。
- 构架选择过多：没有像 C#这样的官方.NET 构架，也没有像 ruby 由于历史较短，构架开发的相对集中。不过这也从另一个侧面说明，Python 比较优秀，吸引的人才多，项目也多。



说明

FLOSS 是基于一个团体分享知识的概念。这也是 Python 能够如此优秀的原因之一——它是由一群希望看到一个更加优秀的 Python 语言程序员加入进行创造并不断改进的团体。

面试试题 2

请简单阐述 Python 语言的应用范围。

解答

Python 作为一门脚本语言，具有广泛的应用，这也是由该门语言的特点所决定的。强大的编程能力、良好的可移植性以及丰富的库都使得 Python 必定具有很广的应用范围。

- 系统编程：Python 提供系统开发 API，能方便进行系统维护和管理。
- 图形处理：有 PIL、Tkinter 等图形库支持，能方便进行图形处理。
- 数学处理：NumPy 扩展提供大量与许多标准数学库的接口。
- 文本处理：Python 提供的 re 模块能支持正则表达式，同时还提供 SGML、XML 分析模块，许多程序员正是利用 Python 进行 XML 程序的开发。
- 数据库编程：程序员可通过遵循 Python DB-API（数据库应用程序编程接口）规范的模块与 Microsoft SQL Server、Oracle、Sybase、DB2、Mysql 等数据库通信。Python 自带有一个 Gadfly 模块，提供了一个完整的 SQL 环境。
- 网络编程：提供丰富的模块支持 Sockets 编程，能方便快速地开发分布式应用程序。作为 Web 应用的开发语言，支持最新的 XML 技术。
- 多媒体应用：Python 的 PyOpenGL 模块封装了“OpenGL 应用程序编程接口”，能进行二维和三维图像处理。另外，PyGame 模块可用于编写游戏软件。

说明

对于 Python 的应用范围，面试者只需要回答出其中关键的几点即可，如文本处理、多媒体应用和网络编程等。Python 在这些方面都有很好的应用。

6.1.2 Python 测试相关

Python 脚本语言有很多支持测试的功能强大的模块，不论是单元测试还是自动化测试都有相应的模块，再加上 Python 的良好跨平台性，所以很多用其他语言编写的程序都会借助 Python 语言进行自动化测试，这样不仅能减少成本，同时还能提高测试的效率。在下一章我们将就 Python 的测试进行详细的讲解，本节只是对 Python 对于测试方面提供的支持进行简单介绍。

面试试题 3

请列举 Python 中被广泛应用的测试框架，并对各种框架做简单介绍。

Offer

解答

Python 主要有 3 种测试框架应用最为广泛。

- zope.testing：zope 项目就是要以一种统一的风格在大型代码中发现和运行测试，zope.testing 支持像 unittest 和 doctest 等传统 Python 测试风格，并且提供了一个强大的分层系统，在这种系统中包含测试的目录可以通过代码设置为层，进而创建一个测试环境。
- py.test：该框架建立了 Python 测试的新标准，目前许多 Python 程序员都会采用该标准进行测试。这个新标准为编写测试提供了符合 Python 风格的习惯做法，使得开发人员可以以非常紧凑而优雅的风格编写测试用例和套件，提高了测试的效率。
- nose：该框架支持和 py.test 同样的测试标准，但却比其更加易于安装和维护。可以说在易用性方面 nose 比 py.test 框架更加优秀。

我们只需对 3 种测试框架进行简单比较，就不难发现 nose 框架的前景更为广阔，其有可能会成为标准。py.test 框架的用户群虽然不大，但习惯了使用这个框架后就很少会改用其他框架。zope.testing 只是在基于 zope 框架的项目受欢迎，所以应用也非常有限。3 种框架都有各自的特点，最好充分了解其特性和差异，根据实际的需要选择适合自己项目的框架。

面试试题 4

请比较 zope.testing、nose 和 py.test3 种测试框架对并行和分布式测试方面的支持。

解答

随着软件项目对代码质量要求的不断提高，测试运行的频率也变得更高，再加上测试驱动开发方法的流行，许多开发人员在开始编写新的代码之前就会预先编写和运行测试，这对于保证代码质量是很有帮助的。

因此，用尽可能多的计算机资源来运行测试是很有益处的，可以大大提高测试运行的效率。对于小的项目使用多个 CPU 就可以解决，但对于更大的项目也许会采用计算机群或专用服务器来运行测试。此时 3 种不同测试框架对于并行和分布式测试的支持的差异就显现出来了。

- zope.testing 命令行支持-j 选项，可以用来指定启动测试进程的个数，而不是在同一个进程中执行所有测试。由于每个进程可以在不同的 CPU 上运行，所以如果在 4CPU 的计算机上运行，使用-j 4 就可以同时使用这 4 个 CPU 来运行所有测试了。
- nose 框架对并行测试的支持在项目 trunk 中得到了实现，具体用法可参考使用文档。
- py.test 框架支持-n 选项，这和 zope.testing 一样可以支持多 CPU 运行测试。另外，该框架还提供了辅助的管理工具，可以在整个测试服务群中分布测试。

通过比较可以发现，py.test 在并行测试方面要优于其他两个框架。py.test 可以同时在多个 Python 解释器版本和多个不同的操作系统上运行测试，如果项目支持多个平台，则 py.test 框架就是一个很好的选择。

Offer

说明

Python 对于测试支持不只是这 3 个测试框架，读者需要对前面提到的 unittest 和 doctest 测试风格有所了解。这样才能更好地进行 Python 测试开发，同时也可以对 Python 测试相关的面试题目能有更充分的准备。

6.2 Python 单元测试

对于 Python 语言的考察，单元测试是一个非常重要的方面。我们都知道 C++的单元测试有专门的 CPPUnit 类来实现，其实 Python 也有其用来做单元测试的模块 PyUnit。如果我们留意各大公司的笔试面试题，可以发现不论公司是招研发人员还是测试人员，都会考查 Python 程序员单元测试方面的知识。通过本节的学习，我们可以对 Python 单元测试从环境的搭建到具体的应用有一个很全面的把握。

6.2.1 Python 单元测试环境搭建

PyUnit 是基于 JUnit 和 XUnit 所开发的，它为 Python 语言提供的测试框架。该测试框架提供了统一而简单的方法来开发 Python 软件测试程序，同时还提供了图形用户界面，用于提高单元测试的可用性。

面试试题 1

你是否用过 PyUnit 进行 Python 代码单元测试？如果用过,请谈谈 Python 单元测试环境的搭建，您可以从 Python 模块安装、测试用例的构建等方面阐述？

解答

对于 PyUnit 单元测试环境的搭建可以从以下两个方面进行解释。

PyUnit 模块的安装需要注意以下两点。

首先，要检测自己所用的 Python 的版本，如果是低于 2.1 版本的，则必须手工安装。

其次，下载 pyunit-1.4.1.tar.gz 模块后，应用命令“tar xzvf pyunit-1.4.1.tar.gz”进行解压，同时确保 unittest.py 和 unittestgui.py 两个文件在 Python 的搜索路径。

说明

本例默认在 Linux 平台使用 Python 语言。

6.2.2 PyUnit 测试框架的应用

在 PyUnit 中，测试用例是由 unittest 模块中的 TestCase 提供的。TestCase 类的实例就是一

个能运行一个单独测试方法的对象。PyUnit 测试框架在运行一个测试用例时，TestCase 子类定义的 setUp()、runTest() 和 tearDown()方法依次被执行，最简单的测试用例只需覆盖 runTest()方法来执行特定的测试代码即可。

通常为了测试的全面性，我们会写很多测试用例，此时就可以使用 setUp()，因为我们只需要在 TestCase 中实现一次 setUp 方法，其他派生类可以从这个子类中继承。如果在测试运行中 setUp 方法出现异常，测试框架会认为测试出错，那么 runTest 就不会再被执行。同样，框架也提供了 tearDown 方法，在 runTest 方法运行后进行回收工作。

从下面的示例代码我们可以对 TestCase 中的 setUp 和 tearDown 方法有更好的理解。这段代码主要是对窗口函数进行测试，包括两个测试用例，主要测试部件默认尺寸和改变尺寸两个函数是否能返回正确值：

```
import unittest
    class SimpleWidgetTestCase(unittest.TestCase):
        def setUp(self):
            self.widget = Widget("The widget")
        def tearDown(self):
            self.widget.dispose()
            self.widget = None
        //测试默认值是否正确
        class DefaultWidgetSizeTestCase(SimpleWidgetTestCase):
            def runTest(self):
            assert self.widget.size() == (50,50)
        //测试改变部件尺寸是否成功
        class WidgetResizeTestCase(SimpleWidgetTestCase):
            def runTest(self):
                self.widget.resize(100,150)
            assert self.widget.size() == (100,150)
```

面试试题 2

请谈一谈对 PyUnit 中 TestSuite（测试套件）的理解。

解答

测试用例会依据所要测试的功能分组。PyUnit 提供测试套件，由 unittest 模块的子类 TestSuite 实现。在每个测试模块中提供一个返回已创建测试套件的可调用对象，会是一个使测试更加敏捷的方法。

测试套件可以嵌套使用，即将很多测试用例的套件放在一起运行。TestSuite 可以像添加一个用例到一个套件一样，将套件添加到另一个套件中。

下面这段代码主要定义了一个测试套件 suite，并且添加了两个测试：

```
//定义测试套件并添加两个测试
def suite():
```

Offer

```
suite = unittest.TestSuite()
suite.addTest(WidgetTestCase("testDefaultSize"))
suite.addTest(WidgetTestCase("testResize"))
return suite
```

面试试题 3

请列举 PyUnit 中将测试代码和源代码分离放置的好处。

解答

在 PyUnit 测试框架中允许将测试用例和测试套件放在所要测试代码所在的模块中，但注意要将测试代码放在单独的模块中。这样分离放置的优点如下：

- 测试代码可以由命令行单独运行。
- 测试代码可以更容易地从交付的代码中分离出来。
- 测试代码应比所要测试的代码改动的频率低。
- 测试代码可以更容易被重构。
- 即使测试策略改变，也没有必要去改变源代码。

面试试题 4

请介绍 PyUnit 测试代码的两种不同运行方式。

解答

通常有两种方法运行测试代码，一种是通过命令行的方式运行，另一种是从图形用户界面运行。

unittest 模块中包含一个 main 方法，可以很方便地将测试模块转变为可运行的测试脚本。main 使用 unittest.TestLoader 类来自动查找加载模块内的测试用例，此时只需将如下代码放在测试模块结尾处即可。当从命令行执行测试模块时，其所包含的所有测试都将被执行，同时可以使用“-h”选项查看所有可用的选项。

另外，PyUnit 还提供了一个图形界面测试脚本 unittestgui.py，将其复制到当前目录后，可以执行 python unittestgui，py 命令启动该测试工具，对 main_runner.py 脚本中所有测试用例进行测试。

面试试题 5

请写一段应用 PyUnit 模块的单元测试代码，要求将所有测试代码都放置在同一个测试套件中。

解答

此时只需将所有测试文件都放在列表中，同时创建测试套件，然后通过循环，将所有列表

Offer

中测试文件执行即可。

说明

本例通常在不同项目中变化会很大，所以用户可以根据自己的需要改动代码，但这个框架是可以重用的。

6.3 py.test 测试框架

py.test 测试框架是一种基于命令行的，用来收集和运行测试的自动化测试工具。它可以在 Linux、Windows 等多种主流操作系统上很好的运行，只要 Python 解释器版本在 2.4 到 2.6 之间即可。该框架具有可扩展的插件结构并且可以直接运行已有的常见 Python 测试套件而无须做任何改动。

py.test 测试框架可以使我们以更符合 Python 的编码习惯来编写测试用例。最奇妙之处在于，如果在测试驱动开发中应用 py.test 测试框架，可以使单元测试的开发开销近乎为零。

Py.test 测试框架可以将单个测试文件分布在多个计算机上运行，这样可以大大提高测试运行的效率，尤其适用于大的项目。同时，该测试框架还具有广泛的应用范围，从具有成千上万的测试用例的大型软件项目到只有几行内联测试语句的命令行脚本均有覆盖。在本章中我们将对该测试框架的常见考点进行分析解释。

6.3.1 py.test 的特点

py.test 作为一种应用广泛的自动化测试工具具有其独到之处。通过以下两道题目我们可以对该框架的特点有一定的了解，若想更好地掌握则需要在实践中慢慢体会。

面试试题 1

请谈谈 py.test 测试化测试框架的特点。

解答

py.test 自动化测试框架的特点如下。

- 成熟的命令行测试工具。
- 自动收集和执行测试。
- 支持多种测试方法。
- 提供无样板模式来配置和部署测试。
- 具有先进的测试跳跃功能。
- 可以通过多 CPU 运行测试以使负载平衡。

Offer

- 具有跨平台运行测试的特点。
- 先进的测试选择和运行模式。
- 具有良好的可扩展性。

说明

对于 py.test 测试框架所提供的的无样板模式，即以 Python 已有的简单函数或方法来配置和部署测试。我们可以通过以下几个方面理解：py.test 可以自动发现测试用例，凡是以 test_*.py 这样方式命名的文件都会被检测到并自动运行；可以通过向测试函数传递不同参数；py.test 框架可以捕获测试输出，包括子进程；允许使用 Python 标准的 assert 断言；具有丰富的错误信息收集功能。

面试试题 2

为什么 py.test 会被广泛应用于测试驱动开发中？

解答

py.test 测试框架被广泛应用于测试驱动开发中。回答这个问题需要从两个方面考虑。首先，我们要先搞清楚什么是测试驱动开发方法。测试驱动开发是软件开发的一种方法，主要特点是在进行编码之前先写好单元测试，这样做的好处有以下几点。

- 编写测试代码意味着我们必须使用所有的可能参数去调用实际并不存在的代码，这就要求我们必须认真考虑编写的代码如何工作以及具备什么样的接口等。
- 可以在编写代码前发现问题，尤其是设计想过的错误。
- 通过编写测试代码我们可以很好地估算实际工作代码的长度。

我们现在已经清楚测试驱动开发的含义，那么再结合 py.test 测试框架的自身特点，这个问题就很好回答了。我们知道 py.test 框架最大的特点就是自动发现和运行测试。这对于测试驱动开发来说是非常有用的，我们只需将测试代码以 test_*方式来对测试文件进行命名，那么所有的测试文件都将会被发现和运行，同时还可以定义运行的顺序。同时 py.test 框架还具有可以传递不通参数给函数的特点。这不但可以增加单元测试的全面性，而且还可以大大降低测试代码编写的开销。所以说，正是因为 py.test 测试框架的自身特点决定了它之所以会被广泛应用于测试驱动开发中。

6.3.2 py.test 框架插件

py.test 测试框架之所有具有强大的功能与其具有的丰富的插件是密不可分的。

面试试题 3

请说出 py.test 测试框架中的插件主要有哪几种。



解答

py.test 测试框架的插件大体可以分为以下几种。

- 先进的 Python 测试插件。
- 测试域相关插件。
- 测试报告和失败日志记录插件。
- 对其他测试框架的支持的插件。
- 主要的调试和帮助插件。

面试试题 4

请列举 py.test 测试框架中关于测试报告和失败日志记录的插件有哪些并对它们的用途进行简单说明。

解答

关于测试报告和失败日志记录的插件主要有以下 4 个。

- pastebin: 将失败或测试会话信息提交给 pastebin 服务。
- xmlresult: 用于生成 xml 报告和持续集成工具 CruiseControl。
- resultlog: 该插件主要提供机读测试结果日志。
- terminal: 执行在终端运行的完整测试的报告。

说明

CruiseControl 简称 CC，是一款由 ThoughtWorks 公司开发的持续集成工具，主要提供了基于版本管理工具（如 CVS）的感知变化或每天定时的持续集成，以及持续集成报告，并以邮件等方式通知相关负责人。

6.3.3 Funcargs 机制

“Funcargs”是“Named parameters of a test function”即测试函数名的缩写。一个“Funcargs”可以是一个复杂对象的一个简单数字。为了使测试函数可以调用每一个参数，需要通过一个工厂函数进行设置。同样也可以通过不同的“Funcargs”来设置参数以重复调用一个测试函数。Funcargs 机制，简单来说就是对测试函数的输入参数进行设置的机制，这是通过 funcarg 工厂函数来实现的。工厂函数接收一个请求对象，该对象可以用来注册安装和卸载函数或访问关于测试的元数据。

面试试题 5

请说出 funcarg 工厂模式的请求对象的属性有哪些。



解答

请求对象会被传递给 funcarg 工厂函数，并且允许其访问测试配置、测试范围和有用的缓存。以下是请求对象属性列表。

- request.function：Python 请求参数的函数对象。
- request.cls：定义了测试函数的类对象。
- request.module：定义了测试函数的模块对象。
- request.config：进入命令行操作和一般配置。
- request.param：查看是否存在通过前一个“metafunc.addcall”函数传递的参数。

6.4 Python 综合测试

Python 脚本语言以其强大的功能被广泛应用于软件开发的各个领域，从软件项目的开发到测试，甚至是测试工具的开发，都少不了 Python 的身影。Python 比其他脚本语言的优势在于其卓越的跨平台性，这对于一些移植项目来说是个非常好的选择。我们可以应用 Python 进行跨平台协作开发，同时应用其跨平台性对软件进行充分的测试。Python 语言作为一门优秀的编程语言，在测试应用上称得上是“长袖善舞”，甚至可以这么说，要想成为一个优秀的测试开发工程师，Python 是必须要掌握的。

现在人们越来越意识到测试的重要性，原因很简单，那就是软件测试的质量直接关系到项目的成败。测试质量越高，也就越能越早地发现问题，从而减少后期解决问题的成本。我们也都听过自动化测试，但在实际应用中却很少应用，甚至说从来就没有应用过。这与测试开发语言的选择有一定的关系。

Python 语言功能强大，可以应用于很多领域的测试：从数据库测试到网页测试，再到性能测试、单元测试。结合 Python 的测试框架，可以将各种测试进行自动化转化。在本节中，我们将就 Python 在测试方面的应用以问题的形式让读者有一个大体的了解。由于篇幅有限，本节只能起到抛砖引玉的作用，想深入了解 Python 在测试方面的强大功能，还需要读者不断深入学习。

6.4.1 Python 数据库测试

Python 作为一种面向对象的编程语言用途非常广泛，其清晰的语法特点和良好的跨平台性使得 Python 在数据库的测试方面也有很好的应用。Python 支持通过引进的自带的 CTYPES 库，可以达到在 Python 脚本中执行已有的动态库中代码的目标。我们就可以通过这个方法对数据库管理系统对外所提供的基于 C/C++接口进行访问。同样，通过使用 Python for .NET 和 Jython 也可以对.NET 的 CLR 代码和 Java 平台的任何类库进行调用。在解决了不同平台间对数据源访问

Offer

的问题之后，对数据库的测试就变得简单了。我们只需要编写一个解释器调用脚本让其作为一个连接上层测试用例和下层数据连接的桥梁，然后通过这个桥梁我们就可以把精力放在对数据库功能和性能的测试用例的编写中，并且这套测试对于任何平台都是适用的。

说明

Ctypes 是 Python 提供的一个用于调用动态链接库和共享库的一个包，我们可以通过这个包来创建、访问和维护 Python 脚本中的 C 数据类型。换句话说，就是以纯 Python 的方式对 C 库文件进行封装。Ctypes 可以在 Windows、Mac OS X、 Linux、Solaris、FreeBSD 以及 OpenBSD 平台工作。

面试试题 1

已知 SQL Server 数据库，如何使用 Python 语言对该数据库进行测试？

解答

想要对 SQL Server 数据库进行测试，我们要先确定 Python 有没有这样的工具对数据库进行操作，具体对于数据库的测试并不难。幸运的是，在 Python 中确实有一个名为 Pymssql 的扩展模块是专门用来访问 SQL Server 数据库的。

Pymssql 模块提供了连接数据库的函数 connect(),该函数允许输入用户名、密码、主机名、数据库以及登录超时等多个参数，同时还提供了很多方法，如提交事务、回滚事务、关闭连接、执行操作及读取结果等。这些方法基本可以满足我们对数据库的操作。

现在再回答这个问题就很简单了。我们只需要根据实际数据库所需要测试的方面进行测试用例的开发即可，而无须考虑与数据库的交互问题了。

6.4.2 Python 网页测试

我们必须清醒地认识到，无论我们的应用程序实现多么完美，在使用时都可能出现错误，尤其是对网页应用程序来说更是如此。即使 Web 应用程序完全正确，但如果应用程序运行在一个有错误的浏览器上或与特定的数据库不兼容，则会有可能出现错误。因此，这些情况都是要在程序发布前进行充分考虑和测试的。

很多时候我们的程序在开发环境可以很好的工作，但一旦部署到一台不同的机器时就会出现很多问题，此时进行充分的安装测试就是非常有必要的。我们需要两种不同层次的测试：高层次的测试包括功能测试、集成测试和安装测试；低层次的测试主要指单元测试。

另外，高层次的测试还包括对用户接口的测试。一般情况下，我们需要进行必要的测试以确保程序的按键和对话框等能正常工作，然后进行测试以确保网络连接、网页服务器、数据库、邮件服务器、应用程序和网页浏览器等能够协同工作。注意：如果我们已经对程序逻辑和各个部件进行了充分的测试，那么我们就不需要过多关注高层次的测试了。

Offer

面试试题 2

假设有一个完整的网页应用程序，有哪些方法可以进行接口测试，如果使用 Python 语言，则需要使用哪个库？

解答

简单来说，对于网页应用程序进行接口测试有两种方法：即困难的办法和简单的办法。

所谓困难的办法就是所有的事情都是手工来做。我们可以使用最熟悉的编程语言中的 Web 库来实现获取和传送请求并验证结果。无疑这样成本是很高的，并且所写的测试受所用语言的 Web 库的限制。

所谓简单的方法就是使用别人已经写好的的工具进行测试。当然，通常来说这些工具也是通过调用一些底层库来工作的。通常来说我们需要更多的定制功能以便更好地进行测试，此时底层库就会变得更加有价值。我们可以根据需求应用底层库来对工具的功能进行完善。

由于用户的操作与网页应用程序的交互是通过 HTTP 协议进行的，所以想要模拟用户的操作只需要给服务器发送等效的 HTTP 请求即可。使用 Python 中的 urllib 库是一个很好的选择，因为它对网页应用程序来说更加简单易读。Python 语言中的 urllib 库就是用来管理与网页的交互的，这个库包括 urllib 和 urllib2。urllib 不需要认证就可以工作，而 urllib2 可以管理基于 cookie 的认证。通常我们会选择 urllib2 库，因为这个库的功能更强大。想要生成获取请求，只需要简单的两行代码即可：

```
>>> from urllib2 import urlopen
>>> page  = urlopen("http://www.example.com")
```

同样，我们可以使用 urlopen 来生成发送请求，此时只需将发送请求字符串作为函数的第二个参数即可：

```
>>> page  = urlopen("http://www.example.com", "query string" )
```

如果要访问的网站需要验证，则事情就会变得更加复杂。但也不要担心，只要安装了 Python2.4 解释器，就可以处理验证的需求了。为了管理基于 cookie 的认证程序，我们还需要使用 urllib2 库中的几个工具，这些工具包括：build_opener、HTTPCookieProcessor 和 Request。build_opener 和 HTTPCookieProcessor 都可以创建一个 opener 对象来管理由网络服务器发送的 cookie，而 Request 对象则主要用来封装请求信息。

说明

urllib2 库中最重要的函数就是 urllib2.urlopen，该函数不仅可以生成获取请求同时也可以生成发送请求，并且当我们试图访问一个并不存在的页面时会返回 urllib2.URLError 出错信息。



面试试题3

请问你是否使用过 twill 工具进行过网页测试，如果有，请谈谈对 twill 的认识并介绍其使用方法。

解答

首先我们来介绍一下 twill 工具。twill 是由 Titus Brown 使用 Python 语言所开发的用于网页应用程序的测试工具。该工具安装简单、功能强大，采用命令的形式和网页进行互操作，使用非常方便。

通过下面代码我们可以很快地掌握 twill 的使用。该例给我们展示了如何使用 twill 的 go 命令来访问特定 HTML 网页，并且使用 show 命令来显示网页内容，代码如下：

```
$ twill-sh
-= Welcome to twill! =-

current page:  *empty page*

>> go http://www.example.com
==> at http://www.example.com

>> show
<HTML>
<HEAD>
  <TITLE>Example Web Page</TITLE>
</HEAD>
<body>
<p>You have reached this web page by typing "example.com",
"example.net",
  or "example.org" into your web browser.</p>
<p>These domain names are reserved for use in documentation and are not availabl for registration.
See <a href="http://www.rfc-editor.org/rfc/rfc2606.txt">RFC
  2606</a>, Section 3.</p>
</BODY>
</HTML>
```

twill 工具的命令都很直观，如 go、show、find、notfind、echo、code、back、reload、agen、follow，通常我们只需要看到这些命令就知道代表什么意思，这样正体现了 twill 工具的易用性。twill 的另一个应用就是对与网页表单的检索和提交，同样也只需使用相关的命令接口。这些命令包括 showforms、formvalue<form_id> <name> <value>、submit<button_id>、formclear<form_id>。其具体用法在此就不做过多的解释了，有兴趣的读者可以阅读 twill 工具的相关文档。

说明

twill 工具最吸引人的地方在于，当我们根据实际需要使用 Python 语言写一些新的命令后，只需将其添加到 commands.py 模块中就可以在 twill 中使用了。

Offer

6.4.3 Python 代码覆盖率测试

对于 Python 程序的代码覆盖率的测试有一个名为 Coverage.py 的工具。该工具可以监测程序以及提示程序已经被执行的部分，然后对数据进行分析以识别那些可以被执行但未被执行的代码。

代码覆盖率通常用来衡量测试的成效，它可以显示代码中哪些部分已经被测试到以及哪些代码没有被执行到。

面试试题 4

请说出使用 Coverage.py 工具进行 Python 代码覆盖率测试有哪些方法。

解答

使用 Coverage.py 工具进行代码覆盖率测试有很多种方法。其中最简单的方法就是使用命令行进行测试。命令行方法可以使我们在运行完测试后得到代码覆盖率。如果读者需要了解项目的代码覆盖率是如何计算出来的以便对项目进行更好的控制，可以参看 Coverage.py 的 API。

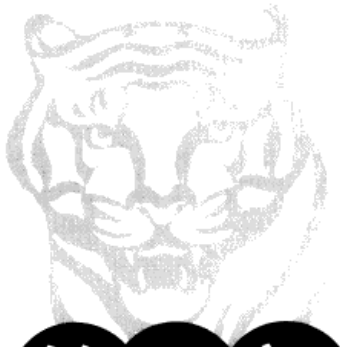
一些测试框架可以提供代码覆盖率测试模块的集成，这样使得在运行测试的同时获得代码覆盖率变得简单，例如 nose 测试框架提高一个 cover 插件。

Coverage.py 工具提供了很多有用的函数，例如 exclude 函数可以直接忽略调代码中我们不感兴趣的部分。html_report 函数可以生成 html 格式的报告。

说明

Coverage.py 工具最初是由 Gareth Rees 设计开发的，但从 2004 年开始就由 Ned Batchelder 对该工具进行维护和扩展了。

Offer



第四部分 测试实战篇

在第三部分测试开发篇中，我们介绍了测试开发的概念和意义，重点讲解 Perl 和 Python 在测试开发中的应用。市面上 Perl 和 Python 的书相当缺乏，相信本书中的面试题和讲解将对大家有所启发。

强调

建议大家学好 Perl 和 Python 这两门语言，它们在测试开发中应用广泛且前景十分广阔。

在本篇中我们将讲解测试实战，也就是项目经验。项目经验是面试考查中的“重头戏”，通常不会出现在笔试中。我们知道，一个有工作经验的测试工程师是否被录取，笔试成绩是次要的——通常及格即可，主要取决于在面试中的表现。其所经历的项目将会被反复诘问，如项目架构、技术特点、模块功能、所遇难点及解法、过程改进、风险控制、测试用例设计、性能分析等都是我们要精心准备的重点。

在这些项目实战中，我们将讲解设计测试用例的思路、如何进行测试开发、如何评估风险以及如何制定测试计划。我们把自己的心得体会融入其中，和大家一起分享。

下面，请大家跟随我进入测试实战篇的天地。

第7章 软件测试面试项目实战

7.1 面试中的项目经验考查

在本章中我们会总结一些经历过的测试项目，并讲述项目经验。如何规划、执行一个测试项目？重点和难点在哪里？如何突破？希望大家能通过本章的学习对实战有个形象的认识。

在面试中，项目经验是必然会被考查的方面，例如：

- 你在这个嵌入式软件项目中遇到的最大的困难是什么？你是如何克服的？
- 在这个金融软件项目中，性能测试是否测出了数据库瓶颈？是如何调优的？
- 在开始这个网站的测试时，你是如何进行风险分析的？
- 你们是如何控制这个手机测试项目的预算的？

有时，一些开放式问题也会被问到，例如：

- 如果你是测试经理，那么当你碰到×××情况时，会怎样调整测试？
- 如果你是测试主管，那么你如何运用 80-20 原则来设计此测试计划以降低风险？
- 如果你是测试经理，那么当你的下属与开发部门发生意见冲突时，你会如何处理？
- 如果你是质量保证经理，那么你将如何保证开发和测试部门都遵守规定的流程？

这些开放式问题实际上也是考查项目经验，因为必须基于一定的项目经验才可能举一反三。因此往往我们需要这么回答：根据我在×××项目上的经验……

如果你没有相关的项目经验，显而易见，你的回答很可能完全抓不住重点，或者基本是纸上谈兵。面试官能提出这样的问题，一定是经历过这种情况并且深思熟虑过。你是否有经验以及想法是否有可操作性，一听就知道了。

面试中的项目经验环节考查是极重要的。不论是天花乱坠的简历，还是笔试的高分，这些都比不上具体工作实践的出色。

引申

有许多资深的 HR 甚至建议取消笔试，而直接面试，主要考查重点之一就是项目经验。



7.1.1 常见考点汇总

- 软件架构描述解释。
- 项目中所遇到的困难及解决方法。
- 项目中的成功及失败经验。
- 测试流程、项目流程。
- 工作任务及业绩。
- 对同类软件的评价与比较。
- 对某种技术的看法。

7.1.2 经典考题例析

面试试题 1

在这个软件项目中你遇到的最大困难是什么？你是如何克服的？

解答举例

这个项目应用到 DB2 数据库。开发人员写了 100 条左右的很长的 SQL 语句来进行查询。每条语句都粘贴在 Word 文档中，至少长达 80 行。在测试时，经理要求我重新实现这些 SQL 查询，以检查开发人员 SQL 查询语句的正确性。这已经超越了白盒测试的要求，不仅是审查代码，甚至相当于重新实现一遍开发过程。

我一边阅读开发人员的 SQL 查询进行分析对比，一边参考需求文档，在二者结果不同时，经常逐条在数据库中比对开发人员 SQL 查询中的限制条件，最终实现了所有的查询，并且找到了相当数量的严重 Bug，例如。

- 逻辑不符合需求。
- 表连接错误。
- 使用了错误的表或字段。
- 限制条件错误或缺少。
- Group by 子句中使用了重名的字段，致使查询无结果。

解析

解决这个问题时不要避重就轻，不要把话题引向非技术方面。如果引向了非技术方面，例如部门之间扯皮严重造成项目延误，给人的感觉毫无疑问是：应聘者的技术水平不高。

回答这个问题，应当提到你能掌握的这个项目中的最难的知识领域，阐述这个领域出现的一个难题。如果面试官根本没有能力听懂，那么更好，你很容易发挥。这是显示你计算机专业水平的一个很好的机会。



面试试题2

你在手机测试项目中的角色是什么？有什么业绩？

解答举例

我们这个测试部门大约有100个人，我在自动化测试组，本组虽然只有4个人，但却是部门中技术水平要求最高的小组。我负责一种测试工具的使用（使用公司另一部门开发的脚本语言撰写测试用例），同时还负责另一种测试工具的开发（使用C++语言）、制订自动化测试的测试计划、分析测试结果、提交测试报告。另外，我还是探索性测试小组的组长，该组由测试专家组成，力图覆盖测试用例之外漏掉的测试。

据统计，自动化测试一共发现了10%左右的软件缺陷，其中我写的脚本测试用例发现的软件缺陷占了其中的70%。另外一种用C++开发的测试工具，在我离开公司时也已试验成功并通过验收。据统计，人工每天8小时内可以测试70条左右的测试用例，而若使用我的工具，在同样的时间内则可以执行300条左右的测试用例，而且它在无人值守状态下可以继续工作。

另外，在探索性测试中，我发现了大量的Dug，在部门每个月的评比中，我基本上都是冠军，并且我报告的有效缺陷数量通常都超过第二名的一倍以上。

解析

这个问题是给面试者加分的问题。面试官有意地让应聘者展示一下业绩。应聘者应实事求是地说明自己的优点和业绩。在回答该问题时，不可过于夸张，否则听起来就过于离奇了。假设把自己的水平“吹”得很高，应聘成功工作后业绩平平，则很打击领导对你的信任。

另外，这个问题一定要集中在技术方面，应避免提到其他方面。例如回答说协调好群众关系或帮助部门获得某个项目，给面试官的印象一定是：此人技术平平，基本没有工作业绩。

7.2 嵌入式软件测试之手机测试实战1

测试计划是测试活动能够在受控状态下顺利进行的重要文档。测试计划文档如何编写也是面试中出现频率较高的问题之一，例如：

- 测试计划文档的内容是什么？
- 测试计划文档何时开始编写？
- 测试计划文档何时编写完成？

本节我们将以手机测试实践为例，对测试计划内容进行分析。限于篇幅，我们的分析将着重于测试的目标与范围、测试的工作量的估算以及测试策略的选取等。

绩效如何评定是很多软件测试部门都要面对的难点问题。很明显，员工的绩效是应当量化可测的，然而如何量化？很多软件测试部门都在进行着摸索和思索。目前，业内还没有统一的

Offer!

原则和标准。这是需要行业共同努力的一个方向。早日制定出科学合理实用的量化绩效的标准，将会大大提高软件测试行业的生产率。在本节，我们对绩效评定的量化方法进行了一些有意义的探讨。

自动化测试工具在手机测试中的应用是手机测试的高端领域。在本节中，我们将以摩托罗拉 Linux-Java 手机测试部门使用的 FlexAuto 工具为例介绍环境配置，举例描述脚本开发方法、要点和原则，同时分析 FlexAuto 工具的局限性，并对几种手机测试工具的优缺点进行对比。

7.2.1 测试计划

事实上，编写测试文档的本身是一个过程，是一个不断做计划、不断修正计划的过程，所以测试计划文档的开始编写时刻就是测试人员加入到软件项目中的时刻，而测试计划文档编写完成的时刻也就是软件产品测试完成并顺利通过测试的时刻。

测试计划文档的内容通常包括。

- 测试的目标与范围。
- 测试的工作量的估算。
- 测试风险的评估。
- 测试策略的选取。
- 各种测试资源的配置和管理。
- 测试产品的跟踪和控制。

测试人员进入项目：

一个手机产品项目在开始之时，先由试产部人员提供产品的市场需求（Marketing Requirement）。例如，这款产品是否带有 500 万像素的相机，是否支持 A-GPS，以及是否支持 WIFI 等功能。

当市场需求文档确定下来后，系统工程师（System Engineer）就开始将市场需求文档中的条目转化为技术需求（Technique Requirement）。例如，使用照相机拍照之后可以被存为哪些格式、存储时间需要多长以及存储位置在哪？又如手机中的视频播放器支持哪些类型的视频文件，支持的最大分辨率是多少，以及支持的视频文件最大文件大小是多少等系统级的参数。

当技术需求文档确定下来后，开发组组长及其组员就开始将技术需求文档中的条目转化为软件需求（Software Requirement）和设计文档。例如，如何实现把照片存档为 JPG 格式，如何分配大分辨率照片的内存，以及如何在规定时间内将内存中的照片存储到闪存中等。同时，用户界面设计工程师也开始设计用户界面和各界面间切换的逻辑等。

说明

在以上 3 个文档的审查会议中，一般都需要有市场需求人员、系统工程师、开发组长及相关开发工程师、测试组长及相关测试工程师、项目经理等的参与。当然，到技术需求文档审查会议阶段，市场需求人员可以是可选的参与者。

Offer

以上都属于手机测试中的需求开发阶段。对于测试团队则是静态测试阶段，即文档测试阶段。在这3个阶段中需要测试人员积极参与，逐步地深入到项目中，了解项目的市场背景、用户群体、系统要求、技术实现。同时，测试人员开始提出问题、从客户的角度对文档进行审查以及查找缺陷等，并在此过程中总结出测试的目标、范围和测试策略等内容。

我们从什么时候开始做测试计划呢？

从加入到项目中的那一刻起就应该开始起草测试计划的工作。

如何确定测试范围？

从上述3个文档审查活动中我们就可以逐步地划分清楚我们所需要的软件测试的范围。我们可以把其范围归纳为如下。

（1）功能性的。

我们可以根据技术需求文档总结出新产品有多少个新的功能模块以及有多少个旧模块（即被选定为新产品的基础软件平台上的已有模块）。

因此这些新的功能模块就被划入功能测试和回归测试的范围，而对于旧的模块，如果存在，就会被划入回归测试的范围。

例如原有的手机平台上已经有手机短信、PIM（Personal Information Management）和收音机等模块，而新的产品要在已有的软件平台上加入 Vcard、SyncML、相机、多媒体播放器等新的功能。这样我们就可以针对技术需求文档中对这些新老模块的划分来确定该项目的功能测试和回归测试的范围。

注意，一个功能模块的细节部分还是必须通过对软件需求和设计文档的分析而得到的。

我们以一个音频播放器为例来确定一个功能模块的测试范围。

- 媒体播放列表

创建新列表

删除列表

插入列表

清空列表

在媒体播放列表中添加歌曲

在媒体播放列表中删除歌曲

在媒体播放列表中改变歌曲排序

在媒体播放列表中插入歌曲

在媒体播放列表中查找歌曲

- 播放控制

顺序播放

随机播放

播放

停止

Offer

暂停

前一首曲目/后一首曲目

播放控制快捷键

音量控制

- 音效控制

各种音效切换

- 与系统中其他发声模块的交互

与电话、短信铃声的交互

与收音机模块的交互

……

- 声道切换

耳机/外放喇叭切换

（2）非功能性的。

非功能性的测试通常属于系统测试的范围，我们可以从技术需求文档中得到这些非功能性的指标。

以下是一个大概的划分。

- 性能测试：列举各个新功能需要测试的性能指标以及其他相关功能和系统级的性能指标。
- 例如在短信的模块中一次性删除 1000 条短信的时间。又如从系统的角度来看手机的待机时间等参数。
- 容错性测试：列举各个新功能在系统出现异常或网络信号中断等情况下的容错处理能力，以及其他相关功能和系统的容错处理能力。
- 例如手机电视在无网络信号时的表现等。
- 兼容性测试：列举各个新功能在与各种常用软件，常用数据格式的兼容能力。
- 例如图片浏览器可浏览的图片格式等。
- 有效性测试（可靠性）：长时间运行新功能模块，不会带来内存泄露、处理器被非正常占用以及系统不会出现死机、崩溃等情况。
- 安全性测试：列举各个新功能对于数据访问权限的控制、数据的加密、数据的存储和恢复等。

如何估算工作量？

通过以上分析，我们确定了一个手机测试项目测试范围，根据这个测试范围我们可以估算出测试的工作量。

下面仍以之前的音频播放器的功能测试为例。

我们首先对产品的音频播放器应用程序进行功能测试的工作总量进行预估。通过以前的工作经验和上一小节中对音频播放器的功能分析可以预估测试人员需要开发 40 个测试用例来覆盖整个音频播放器的功能测试部分。以 40 个测试用例的规模我们预估完成测试的设计和编写需



要 M 天/人。

在测试计划的目标设定环节中，我们会给功能测试或某一个功能的功能测试设定一个通过率。也就是说，只有当功能测试的测试结果高于该通过率时才视为通过了功能测试，否则需要继续挑选测试用例对此轮测试结果中发现的缺陷在新的软件版本上做回归测试。此例中我们以98%为目标通过率。因此，以本例中测试用例总数 40 为例，必须所有测试用例都通过了才可视为该音频播放器应用程序通过了功能测试。

假设根据经验，在该功能测试的项目中需要进行 3 轮测试（两轮回归测试）才能达到。每一轮回归测试的回归率为 75%和 25%，而功能测试期间测试人员需要执行的测试用例的总数为 40×100%+40×75%+40×25%=80。我们以每人每天完成 T 个测试用例为例，预估测试工作量为 80/T 天/人，如表 7-1 所示。

表 7-1 功能测试中的回归测试工作量估计

测试轮次	测试用例总数	测试用例数	回 归 率	测试用例通过数	测试用例通过总比例	目标通过总比例
1	40	40	NA	30	75%	98%
2	40	30	75%	28	95%	98%
3	40	10	25%	10	100%	98%

从上例中可以总结出一个测试工作量预估公式。

$$M+（C+C\times R1+C\times R2+C\times R3+\ldots+C\times Rn）/T$$

其中，M 为设计和完成测试用例文档的时间；

C 为需要测试的测试用例总数；

Rn 为每一轮回归测试的**回归率**；

T 为每人每天完成测试用例的个数。

当然，以上的估算只计算了测试活动中的主体测试用例设计开发和用例执行的工作量，但实际上有以下项目的工作量需要估算。

- 文档审查，包括各个阶段的需求设计文档和测试用例的审查等。
- 测试计划文档的编写、维护。
- 测试用例的设计、编写、维护和自动化测试用例的编写和维护。
- 测试环境的搭建。
- 测试用例的执行。
- 软件缺陷的归档和跟踪。
- 功能测试报告和功能测试报告的编写。
- 其他项目管理的时间，如例会等。

如何制定测试策略？

在测试计划中测试策略是其十分重要的一部分。测试策略覆盖的范围很广，其主要目的就是为了降低测试的风险，在测试的早期针对能够预见的潜在的问题提供解决方案。

Offer

这里我们列举一些常用的测试策略。

（1）基于测试技术的测试策略。

- 使用哪些白盒测试方法对代码进行测试？例如，只对代码进行语句覆盖测试，还是对代码进行全路径测试等。
- 除了对软件的需求文档进行逐条覆盖外，还需要用黑盒测试技术对软件的有效性进行测试。
- 对绝大多数被测组件使用边界值分析法，边界值分析法能有效地查出软件中关于条件和判定的缺陷。
- 对绝大多数被测组件使用等价类划分法可有效地降低测试用例的数量。
- 使用状态变更法可找出设计人员欠考虑的事件/状态变更组合。
- 使用因果图法可解决多组输入/输出组合的情况。

（2）基于自动化测试的测试策略。

- 优先考虑对冒烟测试进行自动化测试。
- 其次是对 BVT（版本验证测试）进行自动化测试。
- 再次是对回归测试进行自动化测试。
- 不对只使用一次的测试用例进行自动化。
- 使用自动化手段对代码进行白盒测试。
- 自动化测试技术的挑选。
- 自动化测试工具的挑选。

（3）基于测试阶段的测试策略。

- 各个测试阶段的质量目标和测试范围。
- 各个测试阶段的进入和退出标准。
- 哪些测试阶段可以进行自动化测试。
- 用户验收测试的组织。

7.2.2 绩效评定

在软件测试部门，尤其是大公司的软件测试部门，有多种工程师分别做不同的工作，例如写测试用例、将用例写成自动化测试脚本、担任小组长、执行黑盒测试以及测试工具开发等，这不同工作的绩效如何评定？如何选出工作优秀者？

这是个相当复杂的问题，笔者记得当年在摩托罗拉 Linux-Java 手机测试部门工作时，领导们也为此事煞费苦心，和同事们讨论了多次。

事实上，基本上所有的软件公司的测试部门都苦于没有良好的绩效评定方法。

在工作中，我总结出了一些想法，在此与大家分享并讨论。

我们将所有的工程师分为 4 组，分别打分，评出各组的优秀员工。

（1）执行测试者：包括用例执行者、探索性测试者和执行自动化测试用例者。

按照发现的每个 Bug 的严重级来进行评分，如表 7-2 所示。



表 7-2 按照发现 Bug 的严重级评分

Bug 的严重级	得 分
1（最高级）	50
2	40
3	10
4	5
5（最低级）	1

注 1：关于严重级定义，可参看如何界定 Bug 的严重级？”。

注 2：如果没有完成测试任务，则每少完成 1 条用例扣除 3 分。

注 3：为了公平起见，每一轮测试后都得交换执行测试者的工作角色。

（2）测试执行小组的各组长。

组长得分是所有组员得分之和的平均值。

每个小组长每月拥有 100 分，当月有效，一般情况下不能自己用，只可以奖给表现优秀的组员，也可以只用其中一部分。

（3）写测试用例者：包括写普通测试用例者、将测试实现为自动化脚本者，如表 7-3 所示。

表 7-3 写测试用例者绩效评分规则表

工作内容	需要满足的条件	得 分
普通测试用例	步骤在 10 步以下	1
	步骤在 11~15 步	2
	使用此用例发现了 Bug	Sev 1（最高级）：25 Sev 2：20 Sev 3：5 Sev 4：1 Sev 5（最低级）：0.5
	未完成工作任务	每差 1 条用例扣 5 分
将用例实现为自动化脚本并调试通过	步骤在 10 步以下	5
	步骤在 11~15 步	10
	未完成工作任务	每差 1 条用例扣 5 分

（4）测试开发工程师绩效评分规则表如表 7-4 所示。

表 7-4 测试开发工程师绩效评分规则表

需要满足的条件	得 分
成功完成一个模块任务	千行代码数*100
程序质量	优秀：千行代码数*100 中等：千行代码数*60 一般：千行代码数*30 不好：千行代码数*（-50）

Offer

续表

需要满足的条件	得　分
代码中被发现有 Bug	Sev 1（最高级）：-25 Sev 2：-20 Sev 3：-5 Sev 4：-1 Sev 5（最低级）：-0.5
修复/处理 Bug	Sev 1（最高级）：-12.5 Sev 2：-10 Sev 3：-2.5 Sev 4：-0.5 Sev 5（最低级）：-0.25
项目延期（当原因是由于测试开发工程师不够努力的情况下）	每人扣 500 分

（5）部门经理拥有推荐权，可以推荐表现优异的员工获得优秀绩效奖。像测试开发小组组长、写测试用例的小组组长等工程师的工作等不容易打分，领导需要特别关注一下他们的工作。

（6）部门绩效可以每个月小评，每个季度中评，每年大评。每次获得优秀绩效奖的员工都应得到物质奖励，如 100 元电话卡，另外还要颁发奖状。优秀绩效奖应当被加总记录，在年末评比唯一的最优员工时使用。也就是说，在年末评比全部门唯一的最优员工时，只看月评、季评的结果。年度最优秀员工计分规则表如表 7-5 所示。

表 7-5　　年度最优秀员工计分规则表

绩　效	得　分
月评优秀	1 分/次
季评优秀	3 分/次

强调

奖状的颁发是非常重要的，因为员工在向他人展示时可以使用奖状来说明，而其他奖品没有标记无法对外证明其功绩。

（7）如果工作有对外联系的内容，还可以请外部客户给各组组长评分，并评价他们的沟通能力与服务水平。

例子

例如使用以上方法为李、赵、田、王 4 个组长评定业绩。各方面业绩评定如表 7-6 所示。

表 7-6　　各方面业绩评定

姓名\项目	除客户打分以外业绩排名	客户评定沟通与服务排名
李	4	2



续表

姓名\项目	除客户打分以外业绩排名	客户评定沟通与服务排名
赵	3	3
田	2	1
王	1	4

最后业绩评定如表 7-7 所示。

表 7-7 最后业绩评定

姓名\业绩评定	分数加总	总 排 名
李	6	A
赵	6	B
田	3	D
王	5	C

注：李排在赵之前是因为领导更重视业务成绩而不是客户反馈。当然，如果领导更重视客户反馈，则李将排在赵之后，这属于领导不同的风格范畴。

如何评比绩效让所有人口服心服，是每个测试部门的领导必须面对的问题。以上的评比方法只是笔者的一些浅见，期盼能引起有识之士的思考，探索出一条业界认同的道路，提升软件测试管理水平，促进软件行业的规范化。

7.2.3 自动化测试工具 FlexAuto

7.2.3.1 FlexAuto 简介

笔者曾经在摩托罗拉 Linux-Java 平台测试部门自动化测试小组负责使用 FlexAuto 实现一些测试用例的自动化测试，也就是将一些手工测试用例改写为 FlexAuto 脚本并调试。

FlexAuto 是摩托罗拉公司自己开发的手机测试工具，通过调用手机软件中的 Agent 来操作手机。FlexAuto 可以从摩托罗拉的测试管理工具 TC 中批量导入测试用例，操作非常方便，如图 7-1 所示。

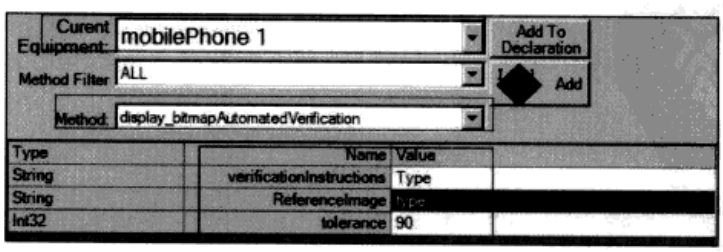


图 7-1 FlexAuto 的运行控制界面

7.2.3.2 FlexAuto 配置和执行

FlexAuto 的配置步骤如下。

Offer

（1）在计算机上打开 FlexAuto 软件，选择 configuration。

（2）添加一部手机：选择 Object -> OK 命令。

（3）IP：用命令行查出手机的实际 IP 地址并输入，此时一般为 192.168.16.2。输入 Pin1 与 Pin2。

（4）在.line1Number 后输入手机号。

（5）在.isRemote 后的下拉菜单中选择 No 命令。

（6）在.learnMenuFilePath 右侧文本框中输入配置文件的地址（如 C:\config\LJ_ Ascension.fam），然后单击 Save 按钮。

（7）关闭再打开 FlexAuto 软件。

（8）在.modelId 右侧文本框中双击，出现下拉菜单，选择手机类型。

（9）单击 Save 按钮。

（10）选中 mobilePhone1 行，即焦点（虚框）停在此行。选择上方的 start，直到 mobilePhone1 行前的黄色变为绿色为止。

- 绿：表示手机连接计算机成功。
- 红：表示手机没连上计算机。
- 灰：表示出问题。

（11）单击 Done 按钮。至此配置完成。

FlexAuto 的执行事项如下。

- FlexAuto 可以顺序执行所有脚本，也可以跳过一部分脚本，用户可以选择。
- 在 FlexAuto 运行前要先指定脚本所在文件夹。
- 可以选择无限时间运行，一轮测试完成又开始一轮。
- 可以指定每两条脚本语句、每两个脚本及每两轮测试之间的时间间隔。
- FlexAuto 提供了 Log 窗口展示即时运行情况。
- 如果不停止程序运行就断开手机与计算机之间的数据线，程序会继续运行。再连接上数据线欲运行 FlexAuto 时，需要重启手机。

7.2.3.3 FlexAuto 的脚本开发

FlexAuto 的脚本使用解释型语言，其语法很简单，用户只要有 C 语言的基础就很容易掌握。FlexAuto 提供了大量 API 函数，测试工程师可以直接使用。

下面我们给出一个实际例子，将一条手动用例实现为自动化脚本。这条用例的目的是验证在“Lock at Power On”这个页面上的默认选项是 “Lock Now”、“Lock at Power On”和“Off”。

脚本程序如下：

```
#########################################################
#
#    Description: UserApps - Regression - Security - Check Phone Lock UI
#
#   Expected Results: Verify preloaded Profiles are in Profile List
```

```
#
#    Test Case Number: LJ.SWTEST.UserApps.Setting:002-002
#
#    Time required to exeute the test case manually : 10 minutes.
#    Time reuqired to automate it : 20 minutes
#
#    Created Date: 06/13/07
#
#    Revision History
#    06/13/07 - bvd367 - Created Script
#
#####################################################
#    Notes: This script is only applicable to LJ6.1 testing.
#####################################################
Required Equipment:
mobilePhone mobilePhone 1
#使用 1 部手机来测试，当使用两部手机来测试时，再加一行:
# mobilePhone mobilePhone 1

Steps:
// 手机回到进入菜单之前的初始界面
mobilePhone 1.keyPress_toIdleDisplay()

// 下面这段程序是命令进入锁屏界面
mobilePhone 1.keyPress_scrollToMenuItem('Security')
#手机进入主菜单，焦点移至 “Security”
mobilePhone 1.keyPress_pushSoftKey('SELECT')
#按下选择键
mobilePhone 1.display_waitForTextType('Phone Lock';60)
#等候 “Phone Lock” 出现在屏幕，若 60 秒内等不到，FlexAuto 报错并终止
mobilePhone 1.keyPress_scrollToTextType('Phone Lock')
#焦点移至 “Phone Lock”
mobilePhone 1.keyPress_pushSoftKey('SELECT')
#按下选择键
mobilePhone 1.display_waitForTextType('Lock Now';60)
#等候 “Lock Now” 出现在屏幕上，若 60 秒内等不到则 FlexAuto 报错并终止

//以下程序段是确定 “Lock at Power On” 这个页面的默认选项是 “Lock 非  //Now”， “Lock at Power On”， “Off”
mobilePhone 1.display_verifyTextTypeOnDisplay('Lock at Power On')
#屏幕上应当出现 “Lock at Power On” ，若没有程序终止
mobilePhone 1.display_verifyTextTypeOnDisplay('Off')
#屏幕上应当出现 “Off” ，若没有程序终止

Cleanup:
//回到初始界面，这一步是为了下一个脚本执行方便
```



mobilePhone 1.keyPress_toIdleDisplay()

在上述例子中，脚本程序分为两大部分：注释和代码。

第一部分是注释部分，包括一些背景信息，例如。

- 测试用例针对模块。
- 手动执行这条测试用例的时间。
- 将这条用例实现为 FlexAuto 脚本所需的时间。
- 开发者。
- 开发时间。
- 测试用例编号。
- 脚本修订创建历史。
- 适用于哪种型号的手机。

第二部分是代码，又分为 3 部分。

- 所需设备。
- 测试步骤。
- 清理。

说明

全局变量在所需设备部分声明，而局部变量在测试步骤中声明。

我们很容易发现，大量的 FlexAuto 自带的 API 函数被用在脚本程序中，例如。

- keyPress_toIdleDisplay()。
- keyPress_pushSoftKey()。
- keyPress_scrollToTextType()。
- display_waitForTextType()。

思索

对于这样一条简单的测试用例，使用人工执行并花不了多少时间。但是，若是集合大批的这种测试用例一起人工执行，就会花费大量的时间和人力。如果用 FlexAuto 来“跑”这些用例，每个用例若没通过就“跑”3 遍，最后只分析没有通过的用例，这样就会大大减少资源消耗。另外，FlexAuto 还可以实现无人值守状态下的运行。

引申

良好的编程习惯是非常重要的，例如清晰简明的注释，脚本开始和最后都让手机屏幕回到初始界面，以及完全严格按照手工用例的步骤没有任何增减等，这些都会带来层次分明、逻辑清楚、顺序鲜明的脚本，便于维护和扩展。

Offer

7.2.3.4 FlexAuto 的局限

笔者认为最大的问题，也是所有自动化测试工具的问题，是机器不能判断千变万化的错误。在这一点上，自动化测试对人工测试的替代有限。

例子

例如运行刚才那条脚本时，屏幕中央忽然出现多道彩线，若是人工测试，很容易就会发现这个问题。可是刚才的自动化脚本是完全判断不了这些彩线的出现的，这条用例仍会通过。而即使在脚本中加入了语句来判断彩线，下次运行时又出现圆斑，但脚本无法判断圆斑的存在，所以仍旧会通过。

引申

即使有的测试工具通过比较两幅图像判断出了圆斑的存在，下次运行时机器忽然发出莫名其妙的噪声，测试工具仍旧判断不出来出现了软件缺陷。

除此之外，FlexAuto 还存在以下一些局限性。

- 不能比较两幅图像。
- 不能判断/测试声音。
- 不能操纵手机关机后的操作。
- FlexAuto 在手机死机时无法重启手机。
- 说明文档更新太慢。
- 不能发送电子邮件。
- 不能重复执行某个脚本。
- 不能自建函数。
- 一些函数完全不能使用。
- 一些函数不能应用于某些类型的手机。
- 如果手机被设置为飞行模式，则 FlexAuto 运行将失常。
- 一些弹出菜单不能被 FlexAuto 读出。
- 一些 Java 应用中的文字不能被 FlexAuto 读出。
- 有时忽然到来的短信会终止掉 FlexAuto 的运行。
- 标志充电完成的响铃有时会终止掉 FlexAuto 的运行。
- 清理部分的代码即使执行失败，执行结果也会被标记为通过。
- 当测试步骤繁杂时，经常会有些 API 函数会失效。
- 当有些步骤的中间间隔较长时，脚本运行失败。
- FlexAuto 的运行过分依赖于界面，因此若界面发生改动，则程序也将不得不改变。而在手机软件开发过程中，界面的改动是很正常的。这样一来，很多脚本会不停地处于维护之中，不能投入大批量测试。

Offer

引申

为了弥补 FlexAuto 的缺陷，后来我们部门又开发了一种工具，剥开手机壳，把电路板连接上导线至一个控制器，在手机屏幕上架设摄像机，将摄像机也连接至控制器，之后将控制器与计算机连接，最后编写脚本来操作每个手机按键和摄像机。

上述工具在使用时需要先手工运行一遍，录制下每一步的手机屏幕，保存下这些图像。运行工具的每一步，手机屏幕也会被录制。该工具会比较两次录制的图像，如果出现不同，就判断为发现手机软件 Bug。

该工具的缺点是。

- 要制造硬件，不容易推广。
- 大量的图像要被保存，占用大量空间。
- 声音仍旧无法被判断/测试。

7.3 嵌入式软件之手机测试实战 2

为了向用户提供更好的产品，就需要在产品发布前进行充分的测试，手机是典型的嵌入式软件产品，而目前对手机的自动化测试程度还不高，绝大部分测试都是手工完成的，既费时又费力。尤其是在集成测试和回归测试两个阶段需要对很多测试用例重复执行。此时如果能借助自动化测试工具，那么就可以极大地降低成本、提高效率。对于需要频繁执行的测试，自动化工具的优势是很明显的。例如在加入新的功能模块后，通常会选取手机各主要功能项进行集成测试，这些测试用例一般是很少变动的，并且执行的频率很高。如果我们能有一个自动化测试工具，只要将这些测试用例手工录入一次，也就是使用工具生成执行脚本，那么在以后的集成测试中，只需要自动执行这些测试用例脚本即可。

在本节中，我们将为大家介绍一款使用 Python 脚本语言所开发的自动化测试工具。在工具的实现中会借助一个手机按键录入执行工具，而 Python 则主要作为连接手机和计算机的桥梁，将整个过程实现自动化，并且对测试结果进行必要的处理。

在本书测试开发篇中，我们已经介绍了 Perl 和 Python 两种脚本语言在自动化测试方面的应用。下面将通过实际项目为大家展示 Python 在软件自动化测试方面的强大功能。

7.3.1 项目需求概述

手机测试质量的高低直接决定了手机产品的市场销路，所以手机生产厂商对于测试是非常重视的。通常来说，各厂商的手机研发中心都拥有庞大的测试团队，其规模甚至会超过软件开发部门。

从单元测试到集成测试、回归测试再到压力测试，每个阶段的测试的通过率都有严格的要求，通过率在不同的项目都有具体的要求。在回归测试中，我们通常会选取一些在新版本依然

Offer

有效的测试用例，以验证软件原有功能在代码修改后是否保持完整。这就意味着，我们在每一次回归测试中都要重复很多测试用例的执行，对一个大的项目来说，在人力物力方面都是很大的考验。如果只依靠手工测试显然效率不够高，此时借助自动化测试工具就可以很好地解决这个问题。

在本项目中开发的工具是基于手机的开发平台，该手机平台支持命令行，并提供了手机各个功能模块的主要接口。同时借助已有的手机按键录制和执行工具 ATA，在使用 Python 脚本工具把整个过程自动化。该工具所要实现的功能如下。

- 在第一次执行手工测试的同时录制测试脚本。
- 使用自动化测试工具作为手机和计算机的桥梁。
- 在脚本录制完成后，将脚本传输到计算机中。
- 自动执行已经录制好的测试脚本。
- 对测试结果进行处理，并生成测试报告。
- 支持测试脚本批量执行。

说明

该工具也有不足之处，那就是不能完全实现自动化，仍需要手工的干预，这也是由所测产品的局限性决定的。因为对手机功能的每一个测试用例都有自己的测试点，需要执行不同的操作，所以这就需要进行一次手工测试，在测试的过程中通过工具来录制脚本。

7.3.2 PyATA 自动化测试工具

自动化测试工具 PyATA 是基于已有的手机按键操作录入执行工具 ATA（Application Test Assistant）所开发的。为了更好地了解该工具，我们需要先了解 ATA 的工作原理和使用方法。

下面先了解一下 ATA 的工作原理。ATA 是应用测试助手的简写，它是使用 C 语言开发的、运行于特定手机平台的一个测试工具。它可以记录鼠标和键盘事件并且将其翻译成简单的脚本，然后可以通过执行该脚本来驱动手机对外提供的用户接口。

ATA 工具的开发有如下的目的。

- 提高测试覆盖率（尤其是应用于性能测试方面）。
- 可以节省在测试方面所花的人工成本。
- 可以捕获很难由手工复现的严重错误。

ATA 使用非常方便，我们可以在执行手工操作的同时运行这个工具，而在手工测试结束后，其就会自动生成可执行的脚本。脚本的执行不受环境的影响，可以全天执行，这对于性能测试是非常有用的。另外，ATA 还支持脚本的录制和回放。对于测试脚本的回放和录制一样，都是通过命令行的形式在手机平台上执行的。

说明

关于 ATA 这个工具我们不需要了解太多，只需要知道如何使用即可。该工具的基本原理

Offer

就是基于 QT 库所实现的录制回放工具。目前各大手机研发机构都有功能类似的工具，当然各种工具的实现也都各不相同。我们这个项目主要介绍如何借助 Python 脚本语言将脚本执行自动化。

虽然 ATA 功能很强大，但也有不足之处，那就是每次在录制测试脚本时需要执行多个命令，也就是说没有连续性。同时，该工具将测试脚本放置于手机内存中，并将运行结果也保存在手机中，这样就会受手机的性能和存储空间的限制。还有一个问题是，对于测试脚本的执行，不能根据需要进行选择性执行，同时测试结果也不能做必要的处理。

此时我们会想，如果有办法能把手机和计算机连接起来，将测试脚本和执行结果都保存在计算机中，并且通过计算机对测试结果做处理，就会弥补这个工具的不足之处了。

对于上述问题是有办法解决的，这就是我们在本项目中所要为大家提供的解决方案：PyATA 自动化测试工具。

我们的想法其实很简单，就是借助 Python 语言强大的跨平台能力和对文件的处理能力，将脚本的录制过程半自动化；而一旦录制好脚本，就可以将脚本的执行过程自动化了。其中脚本的录制过程必须要有人工的干预，这是由手机测试用例的独特性所决定的：每一个测试都需要不同的操作，且必须先手工进行，然后将整个执行过程通过 ATA 录制成脚本。

为了能执行 Python 程序，则需要在本地计算机安装 Python 解释器。我们所使用的手机平台为 Linx-Java，该平台支持命令行操作，只需要通过计算机远程登录手机固定 IP“192.168.16.2”就可以对手机进行交互操作了。

PyATA 自动化测试工具主要有以下 4 部分组成。

- 远程登录手机。
- 录制测试脚本。
- 执行测试脚本。
- 处理测试结果。

接下来我们会给出每个部分的关键代码，以便读者能更好地进行理解。我们主要学习的是这种处理问题的方法，看我们是如何借助 Python 脚本的粘合剂功能将不同的硬件软件联合起来协同工作。

为了连接本地计算机和手机，我们需要使用 Python 库中的 telnetlib 模块，该模块主要用于处理远程连接，只需要给出 HOST 即可，这里就是刚才所说的手机 IP：“192.168.16.2”。主要代码实现如下：

```
HOST = "192.168.16.2"
tn = telnetlib.Telnet(HOST)      //连接手机
tn.read_until("login: ")         //等待登录提示
tn.write("root\n")               //使用 root 用户进行登录
print "Login with root!"         //打印登录信息
```

接下来我们需要使用 ATA 工具进行脚本的录制。ATA 工具通常会内置于手机的内存中的特定文件夹中，在使用命令：./atadaemon –R 进行录制测试脚本前需要先执行配置环境变量的 shell

脚本。在录制结束后需要运行命令：./atadaemon –H 来停止脚本的录制。对于已经录制好的脚本执行命令为：./atadaemon –R。

下面来看一下已经录制好的脚本，该脚本所要做的测试就是当选中“Email”时，事件“KeyPress”将会被执行：

```
[Action]
ApName=messaging      //Ap名称，为信息
Action=KeyPress       //操作是按确认键
Index=3
Text=Email
ResID=TXT_RID_LIST_A_EMAI
Event=
Key unicode:0,keycode:4115,modifiers:0,ispress:1,isautorepeat:0,time:2627762;
Key unicode:0,keycode:4115,modifiers:0,ispress:0,isautorepeat:0,time:2627923;
[End]
```

在测试脚本录制完成后，我们可以使用 Python 中的 tarfile 模块，将所有脚本打包，然后将其传送到计算机中，代码如下：

```
//将所有录制好的脚本打包
tn.write("mkdir ATA_Scirpts\n")
tn.write("cp case*.txt ATA_Scirpts\n")
tn.write("busybox tar -cvf ATA_Scripts.tar ATA_Scripts/\n")

//将所有脚本从手机特定目录中取出
#confirm the full path of get_file.exe
localDir =os.path.dirname("get_file.exe")
curpath=os.path.normpath(os.path.join(os.getcwd(),localDir))
now = re.sub('\\\\',r'\\\\',curpath)
nowpath = now + '\\\\' +"get_file.exe"
tar_name = "ATA_Scripts.tar"

//通过执行程序 get_file.exe 将脚本发送至计算机
cmd_name = "get_file.exe " + tar_name
os.spawnv(os.P_NOWAIT,nowpath, [cmd_name])
tn.write("cat ATA_Scripts.tar | busybox nc 192.168.16.1 1704\n")
tn.write("exit\n")
```

由于 ATA 支持对测试脚本列表的执行，所以将脚本录制好以后放在本地计算机，只需要通过数据线将手机和计算机连接，然后使用 PyATA 将所要执行的测试脚本列表执行即可，此时完全是自动化的操作了，不再需要人工的干预。对于一些数量较大的测试列表，我们可以在下班前运行 PyATA 工具，然后在第二天早上到公司时收到测试结果报告。这对于性能测试是一个很好的选择。

说明

PyATA 作为手机测试自动化工具已经得到了具体的应用，总体来说是非常好的。但有时

Offer

会受手机硬件的制约，有可能会在自动执行几个小时测试后出现死机的现象。

7.4 网络测试实战——数据传输

本节我们将以金融软件为例来讲述有关网络数据传输方面的测试。

金融软件通常需要进行大量的实时数据传输，其运行的整个过程就是对所有的交易所（如股票、证券、期货、外汇等）的实时交易信息进行采集，然后通过相关软件对其进行分类，并通过服务提供商的主干网传输，最后实时分发到世界各地。

现有的新的分发模式对其进行了优化，使得本地采集的数据能够在很短的时间内在本地进行分发，然而这些数据还是会被传输到主干网上进行分发。在这样的一个完整的过程中必定存在信息的打包、传输、解包和信息的存储、操作、检索的过程。

7.4.1 金融软件特性和测试重点

金融软件通常是被作为金融产品提供商的一种服务而存在的，有时甚至作为独立的服务提供商存在。它有以下几个基本特征。

- 传输数据准确。

数据准确是最为关键的。因为金融软件中的数据绝大多数都涉及到钱。在这一点上再小的差错也会给金融服务提供商带来巨大的麻烦。

- 传输速度快。

信息传输速度是金融软件重用性能指标之一。随着科技的发展，我们能感受到金融服务的效率也在不断的提高。

例子

前些年在银行跨区域汇款都需要隔日到账，而现在几乎是实时到账。因为以前银行的数据中心处理速度有限，像跨区域汇款类似的操作首先会被下级的数据中心记录，然后在晚上再由总的数据中心进行同步。

再如股票信息服务，在一百多年前跨区域的股票信息服务都是靠服务提供商进行飞鸽传输实现的。而后则是电报，此后才有了计算机网络传输。现如今对一条股票信息产生、传输到分发给世界各地的用户通常只有几十至几百毫秒。而只分发给区域内（例如城际、省际）用户甚至能达到个位数的毫秒级。

- 系统稳定。

即保证软件能稳定地、持续不断地将信息传输到客户端。这需要金融软件能够克服各种极端或异常情况，例如交易量突然骤涨、传输数据不符合规范、断电、断网线，甚至大到地震、台风、战争……这都是金融服务提供商应事先考虑到的。

- 数据的增值业务。

实时的数据通常是生硬的，它给予客户的通常是该金融产品当时的状态。而客户需要做出更多的判断时还需要金融产品的历史数据、统计指标等。这就需要金融服务提供商提供更多的有价值的数据增值服务以吸引客户。

数据的增值服务，特别是历史数据的统计准确性是金融软件测试的一个难点。虽然每个数据指标都有一个公认的算法，但是因为各个金融服务公司实现的方式或理解不同得到的结果也会有一定的差异。因此，测试人员需要具备一定的金融知识，而且还少不了专业的金融人员配合。

因此，针对上述的几个关于金融软件的特征，软件测试的开展通常会包括以下几个方面：

- 针对数据的准确性。

在每一个传输的环节都要对输入的数据和输出的数据进行解包，以确定数据经过协议转换、传输后未出现数据字段值、数据类型、数据数量的不一致。此外还应该对最终用户收到的数据进行检查比对。

这类数据检查的工作量巨大，适于使用自动化测试工具来替代人工操作，这同时保障了正确率。具体做法是：在数据的输入端 VLAN 和输出端 VLAN 上用数据包抓取工具来分别抓取数据的输入包和输出包然后将其解包，并比对输入、输出数据的字段值、数据类型、数据数量的一致性。

- 针对数据的传输速度。

数据在经过金融软件的处理和网络时都会产生时延。在软件测试中可以通过对每个传输环节的软件处理速度和网络传输速度进行测定以确定每个环节是否达到标准，并可以确定整个数据传输系统的瓶颈。另外，还可以通过测试数据从采集阶段产生、经处理、网络的传输、最后到达客户终端的总延迟以获取数据传输系统的总的性能概况。

这类检查工作的工作量同样巨大，适于使用自动化测试工具来替代人工操作，这同时也保障了正确率。通常做法是在数据的输入端 VLAN 和输出端 VLAN 上用数据包抓取工具来分别抓取数据的输入包和输出包，然后将其解包，并比对输入、输出数据被发出和收到时间以确定其平均时延、最短时延和最长时延等指标。以上方法是在数据进行过协议转换时的必然方法，如果数据在经过处理和传输后并没有协议格式，甚至没有内容上的改变（在这种情况下，通常数据会先由协议 A 转换成为协议 B 以便在某个网络上传播，此后又在接收点被转回协议 A），那么我们抓取输入、输出两端的数据进行比较就更有可靠性了。

- 针对系统的稳定性。

在基本功能测试完成的前提下，对软件进行压力测试以确定软件可承受的性能极限，再以此确定期望的正常工作性能值（通常为性能极限的三分之二）。此后测试人员可以依照这个指标对软件进行浸透测试（Soak Test），以确定软件在确定期望的正常工作性能值下能长时间的正常工作。

当正常工作性能值被确定之后，测试人员就可以测试软件在正常工作性能值下处理各种异常的能力，如短线、断电及输入数据量异常等情况。



- 针对数据增值服务。

这一方面的软件测试通常属于功能性测试的范畴，需要测试人员具备一定的金融知识，而且还需要专业的金融人员配合。

7.4.2 测试活动的开展

网络数据传输类的软件测试同样要经历测试计划、测试环境搭建、测试用例编写、测试和测试报告等阶段。与其他测试不大相同的是其测试环境搭建阶段的工作量较大、涉及面较广。在一般情况下，网络数据传输类的软件需要将同样的复制安装在两台甚至是多台同样的服务器上，以实现工作机与备用机在出现异常时的自动切换，这可以避免因为某一台机器产生断网、断电或宕机时导致整个系统无法正常工作。这样的软件有效性指标通常是 99.95%，即整个系统 1 年之内无法正常工作的时间不得多于 5 分钟。

我们需要安装和搭建多台软硬件环境相同的服务器，其软件环境通常包括操作系统、杀毒软件和软件运行状态监控终端等。

此外，测试人员还需要将服务器接入用于测试的数据源。这样的数据源有时是将真实的数据引入测试网络，例如使用 WireShark、TCPDump 等软件在真实数据网络中抓取并筛选数据包，这有利于测试一些验证功能正确性的测试用例。但是如果需要提高软件正向（Positive）测试用例的覆盖率，和软件产品对一些异常情况的处理能力、恢复能力的覆盖，那就需要测试人员自行设计测试数据，并制作有效的测试数据，再使用数据包发送设备将设计好的数据发送给被测软件。

说明

常用于发送数据包的软件有 TCPReplay。它是一个基于 Linux 平台的软件，支持相当高的发送速率（能达到十万个包/秒），但是速率不是非常稳定。软件自身提供的即时发送速率统计与实际速率有一定误差。

对于测试数据的设计，常用到的方法有等价类划分、边界值、语法测试等方法。这些方法对测试网络协议中的数据字段都有很好的针对性。

为了对比输入和输出数据的一致性和获取数据的时延，我们还需要一台服务器来连接数据输入端的 VLAN 和数据输出端的 VLAN。在这台服务器上通常需要两块网卡，一块连接数据输入端的 VLAN，另一块连接输出端的 VLAN，然后同时抓取数据进行比较，以获得数据的时延和输入和输出数据的一致性。

除此之外，测试人员可能还需要对数据传输的网络环境进行模拟，此时可以通过在数据传输的网络上添加一台网络环境模拟器来实现。网络环境模拟器至少需要两块网卡（一台被测试服务器对应一对输入/输出网卡），其中的一块网卡接入输入的数据，该数据进入模拟器后通过网络环境模拟软件，然后再由另一块网卡发送到数据的输出端。当数据进入模拟器后通过网络环境模拟软件时，模拟器可以模拟网络的时延、掉包率、误码率和带宽等环境，这样就可以测

Offer

试软件在各种网络环境下的表现，如图 7-2 所示。

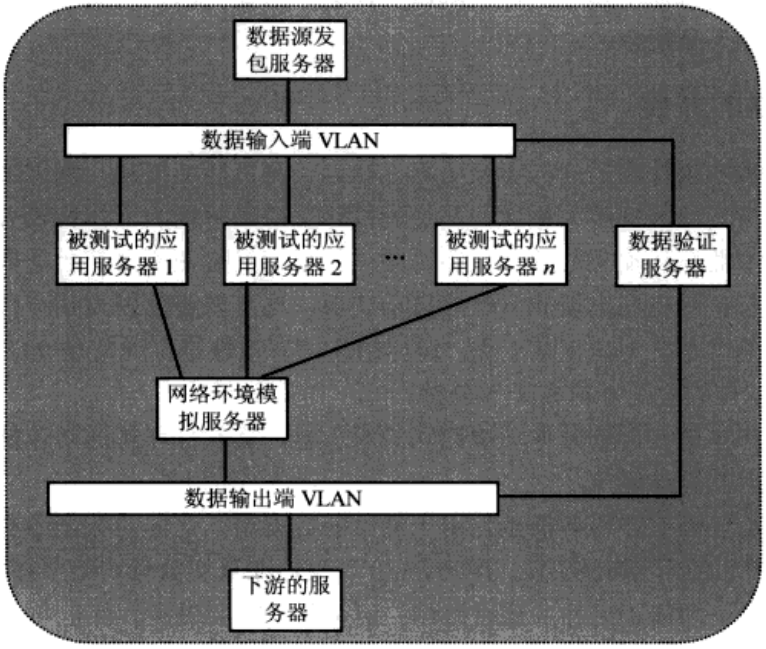


图 7-2 网络数据传输测试

7.4.3 网络数据传输类的软件测试阶段

网络数据传输类的软件测试的测试阶段与其他软件的测试阶段大同小异，它们同样经历单元测试、集成测试、系统测试和用户验收等阶段，其集成测试和系统测试通常都在一个自行搭建的环境中完成。其数据源通常为数据发送软件，而其数据通常为测试人员自行设计的数据。性能测试是该类软件系统测试中需要重点进行的活动之一。

在用户验收测试阶段，软件生产企业通常会有一个长期运行的数据传输系统的模拟系统，其中所传输的数据就是从真实数据源接入的数据。这个环境甚至可以被看作是生产环境中的数据传输系统的备份，这样就很好地模拟了被测软件的最终使用环境。于是，原有系统中的某个软件的替代品（被测软件）就被接入系统中替代原有的老产品（模块）进行测试。当被测试软件（模块）通过该阶段的测试后，该软件（模块）就可以在生产环境中的数据传输系统中正式替代老软件（模块）。

7.4.4 网络数据传输类的软件测试自动化

网络数据传输类的软件测试测试环境的特征就是需要搭建多台各种功能的服务器。因此在进行自动化测试时需要经由一台服务器对多台服务器进行远程控制来完成测试，最终收集测试结果（远程服务器上的性能日志文件、软件的统计日志和事件日志等）。网络数据传输的自动化

Offer

测试如图 7-3 所示。

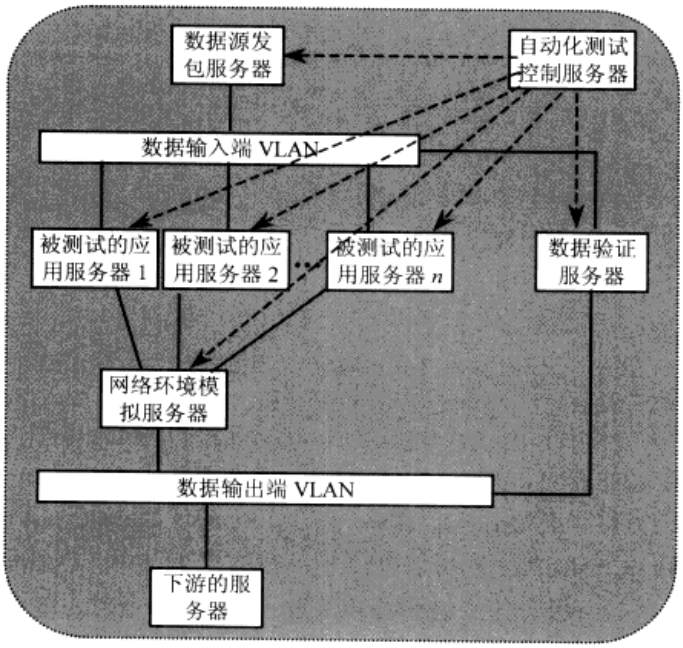


图 7-3 网络数据传输的自动化测试

例如在 Linux 与 Linux 服务器以及 Linux 与 Windows 服务器之间经常会用到的 Telnet、SSH（Secure Shell）、FTP、Samba 等技术都可以实现发送命令或文件传输。

又如在 Windows 与 Windows 服务器之间，可以使用 Windows 自带的 WshRemote 模块。以下代码是使“test.js” 文件在“remoteserver”上远程执行：

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
RemoteScript.Execute

Do While RemoteScript.Status <> 2
WScript.Sleep 100

Loop
```

另外，还可以通过 Windows 自带的 WMI 等模块来实现对远程 Windows 服务器进行管理。例如，可以使用以下代码来查找并关掉在 strComputer 上运行的“Traffic.exe”进程：

```
Option Explicit
dim strComputer, objWMIService, colProcessList, objProcess,

strComputer = "192.168.0.1"
      Set objWMIService = GetObject("winmgmts:"&"{impersonationLevel=impersonate}!\\" & strComputer &
```

```
"\root\cimv2")
        Set colProcessList = objWMIService.ExecQuery ("SELECT * FROM Win32_Process WHERE Name =
'Traffic.exe'")
        do while colProcessList.count>0
            set colProcessList = objWMIService.ExecQuery ("SELECT * FROM Win32_Process WHERE Name =
'Traffic.exe'")
            If colProcessList.count>0 Then
                For Each objProcess in colProcessList
                  objProcess.Terminate()
                Next
            End If
        Wscript.Sleep(1000)
            set colProcessList = objWMIService.ExecQuery ("SELECT * FROM Win32_Process WHERE Name =
'Traffic.exe'")
        Loop
```

这些方法都可以通过各种语言脚本来直接调用，如 VBScript、JavaScript 及 Perl 等。

此外，Windows 还提供了图形化的远程访问方式 Remote Desktop Connection。我们在开始菜单的 Run 命令框中输入 mstsc.exe 就可以使用它。在这个工具中，Option-〉Program 选项中有一个 Start a Program 选项，当用户勾上这个选项并输入 DOS 命令时，该命令就会在登录远程服务器时在远程的服务器上执行；同时如果勾上 Drivers 共享选项，还可以实现远程的文件共享。这就大大地方便了对软件运行日志的收集。

我们可以利用 Windows 提供的 Remote Desktop Connection 的 API（AxInterop.MSTSCLib.dll）进行编程，以实现可视化的远程控制。如以下 C#代码就是实现在 panel1 上动态的加载一个 tempAxMsRdpClient6 （Remote Desktop 实例）实例，并在连接远端服务器时打开远端服务器（192.168.0.1）上的“calc.exe”（计算器）应用：

```
    AxMSTSCLib.AxMsRdpClient6 tempAxMsRdpClient6 = new AxMSTSCLib.AxMsRdpClient6();
    System.ComponentModel.ComponentResourceManager resources1 = new System.ComponentModel.Component
ResourceManager(typeof(RemoteDesk));

    ///BeginInit
    ((System.ComponentModel.ISupportInitialize)(tempAxMsRdpClient6)).BeginInit();
    tempAxMsRdpClient6.Dock = System.Windows.Forms.DockStyle.Fill;
    tempAxMsRdpClient6.Enabled = true;
    tempAxMsRdpClient6.Location = new System.Drawing.Point(0, 0);
    tempAxMsRdpClient6.Name = "ServerName";
    tempAxMsRdpClient6.OcxState = ((System.Windows.Forms.AxHost.State)(resources1.GetObject("axMsRdpClient41.
OcxState")));
    tempAxMsRdpClient6.Size = new System.Drawing.Size(1100, 912);
    tempAxMsRdpClient6.TabIndex = 0;
    this.panel1.Controls.Add(tempAxMsRdpClient6);
    ((System.ComponentModel.ISupportInitialize)(tempAxMsRdpClient6)).EndInit();
    ///EndInit
```



```
tempAxMsRdpClient6.Focus();
tempAxMsRdpClient6.Server = "192.168.0.1";
tempAxMsRdpClient6.AdvancedSettings3.RDPPort = "3389";
tempAxMsRdpClient6.UserName ="UserName";
tempAxMsRdpClient6.AdvancedSettings3.ClearTextPassword = "Password";
tempAxMsRdpClient6.AdvancedSettings3.RedirectDrives = True;
tempAxMsRdpClient6.SecuredSettings2.StartProgram = "clac.exe";
//set the start up command

tempAxMsRdpClient6.Connect();
//connect
```

7.5 网站测试实战

在汤森路透公司媒体部，笔者的工作任务是测试公司的系列金融网站（如路透中文网）。网站中有很多个模块，涉及到很多的部门，我测试的范围是网站的一小部分。

在汤森路透北京研发中心，测试工程师被称为 QA（Quality Assurance）。然而，大多数公司里的 QA，实际上不是测试工程师（尽管多半是测试工程师出身）。一般来说，QA 是协调开发、测试、设计等部门完成项目的质量保证工程师，他们往往具有丰富的软件项目流程知识、丰富的软件测试经验，还精通 CMMI 等各种认证。

媒体部的 QA 部门，领导和几个工程师在美国，在印度有一批外包员工，还有就是在中国。中国和美国的都是正式员工。目前在北京的测试工程师只有我们两个人，正在加速扩大。

驻守在美国的直属领导是一个印度人，他每周和我们开两个电话会议，布置任务与讨论问题。在中国当地的领导，理论上不直接给我们分派任务，但是每周也要和我们开会，了解并指导工作。这样的话，我们实际上有两个领导，两个领导的指令都要听。

说明

大公司经常是这样，一个部门的人在几个国家（地区），驻地领导和直属领导都会指导你的工作。很多时候，你的理论上的直属领导只是象征性的，反而驻地领导实际上给你分派任务。

在此期间，我制订过完整的测试计划，做过安全测试、白盒测试、性能测试及灰盒测试等，有一些自己的体会，在这里愿与大家分享。

当然，凡是属于公司内部机密信息的，细节省略，请大家谅解。

7.5.1 Jmeter 性能测试

7.5.1.1 Jmeter 原理、插件及测试步骤

我们先后使用 Jmeter 2.3.1 和 Jmeter 2.3.2 来进行性能测试。之所以这样选择，主要原因是这是开源免费的软件。Jmeter 的原理与 Loadrunner 相同，都是通过中间代理、监控和收集并发

Offer

客户端发现的指令，并把它们生成脚本，再发送到应用服务器，然后监控服务器反馈的结果的一个过程。

在多种操作系统上，我们主要进行了并发用户测试、疲劳强度和大数据量测试。

此时需要下载一个工具：Badboy，利用该工具可以录制操作，然后选择将脚本保存为 Jmeter 脚本，最后利用 Jmeter 可以打开并修改脚本。

部门同事为 Jmeter 开发了两个插件。

（1）JVM Counter：可以监控到 Java 服务器的 JVM 上的每个应用的内存的使用情况，但它无法监控到 CPU 的情况。

（2）SNMP Checker：实时监控服务器的整个操作系统性能，包括 CPU 和各个应用的内存的使用情况、Disk I/O 等，例如 JVM 这个应用的内存使用。理论上它可以监控所有提供 SNMP 协议监控的对象，如 Linux 服务器、MySQL 等。

SNMP Checker 是 JVM Counter 的加强版。JVM Counter 只能监控服务器上的 Java 运行环境 JVM，而 SNMP Checker 不但能监控非 Java 服务器，而且能监控服务器上整个操作系统的性能。

性能测试的步骤如下。

（1）测试计划与场景制订。

（2）使用 Badboy 录制脚本，然后将脚本转换成 Jmeter 可使用的文件。

（3）在 Jmeter 中修订脚本（边执行边修订、经常需要检查 Jmeter 日志来判断执行是否正确），如进行参数化等，如图 7-4 所示。

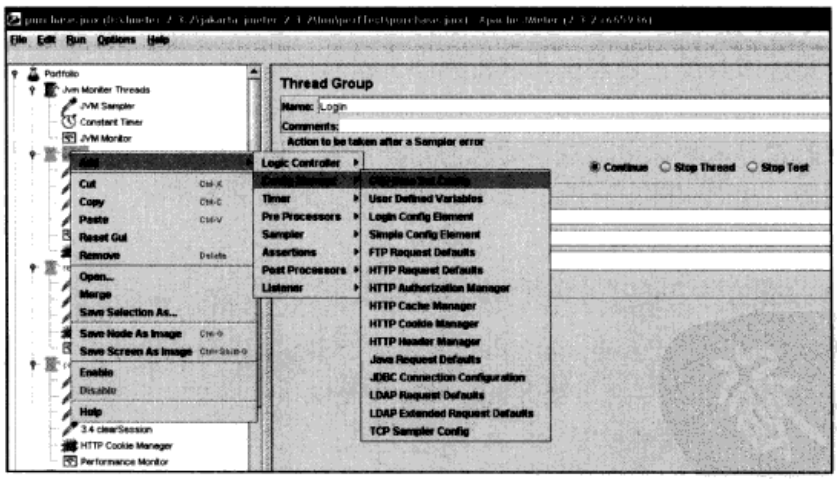


图 7-4 在项目中加入一个参数化文件

（4）执行测试。

（5）分析结果。

7.5.1.2 参数化步骤

参数化的方法有以下 3 种。

Offer

（1）选择 Options→Function Helper Dialog，选择“_Random”函数。

（2）选择 Options→Function Helper Dialog，选择“_StringFromFile”函数。

（3）利用 CSV Data Set Config。

这里介绍第 3 种方法。其参数化步骤如下。

（1）建立文件，假设为 buySearchKeyword.txt，内容为：

```
ibm.n,
msft,
google,
```

（2）将此文件与 Jmeter 脚本文件(.jmx)放入同一文件夹中，例如在我的机器上是 D:\Jmeter-2.3.2\jakarta-jmeter-2.3.2\bin\perfTest\。

（3）在 Jmeter 项目中添加一个 Config Element: CSV Data Set Config，输入如图 7-5 所示的内容。

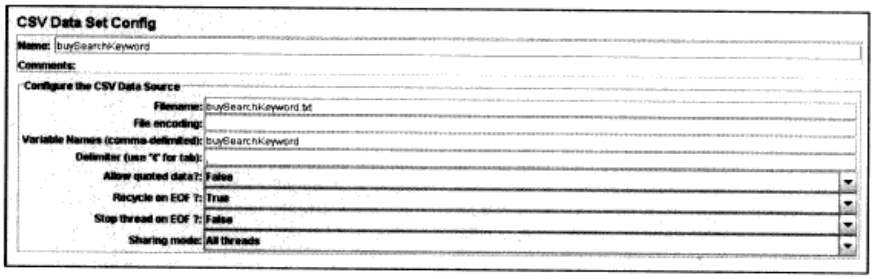


图 7-5 在 CSV Data Set Config 中输入内容

（4）在需要参数化的地方进行参数化替换。在图 7-6 中，Search 的值被替代为${buySearchKeyword}，变量名是 buySearchKeyword，这里的变量名要与步骤(3)中的 Variable Names(comma-delimited)文本框中输入的变量名保持一致。

7.5.1.3 Jmeter 局限

笔者总结了一些 Jmeter 的局限，如下所示。

- Jmeter 的报表较少，提供的信息不足。例如 Jmeter 没有 LoadRunner 那样的网页细分图，因此在主要时间损耗的细节定位上无能为力，没有第一次缓冲时间图，同时也没有网络时间与服务器时间的对比。
- 不能编写脚本程序，而采用图形化的替代方式，看上去似乎简单，实则经常复杂化了问题。
- 图表中不能放大细节。
- 运行机制似乎存在问题（例如参数文件上有的行会被 Jmeter 漏读，正确情况下每一行都应当被读出）。
- Jmeter 没有 IP 欺骗功能，IP 欺骗是指在同一台计算机上模拟出多个 IP 地址来分配给并发用户。
- Jmeter 主要是通过增加线程组的数目或是设置循环次数来增加并发用户，而 LoadRunner 可以通过在场景中选择要设置什么样的场景，然后选择虚拟用户数。Jmeter 不能在不同

Offer

的代理上设置不同数目的虚拟用户。

- Jmeter 使用的人较少，网络上相关资料不全面，需要自己去揣摩。
- Jmeter 支持的协议远远少于 LoadRunner。

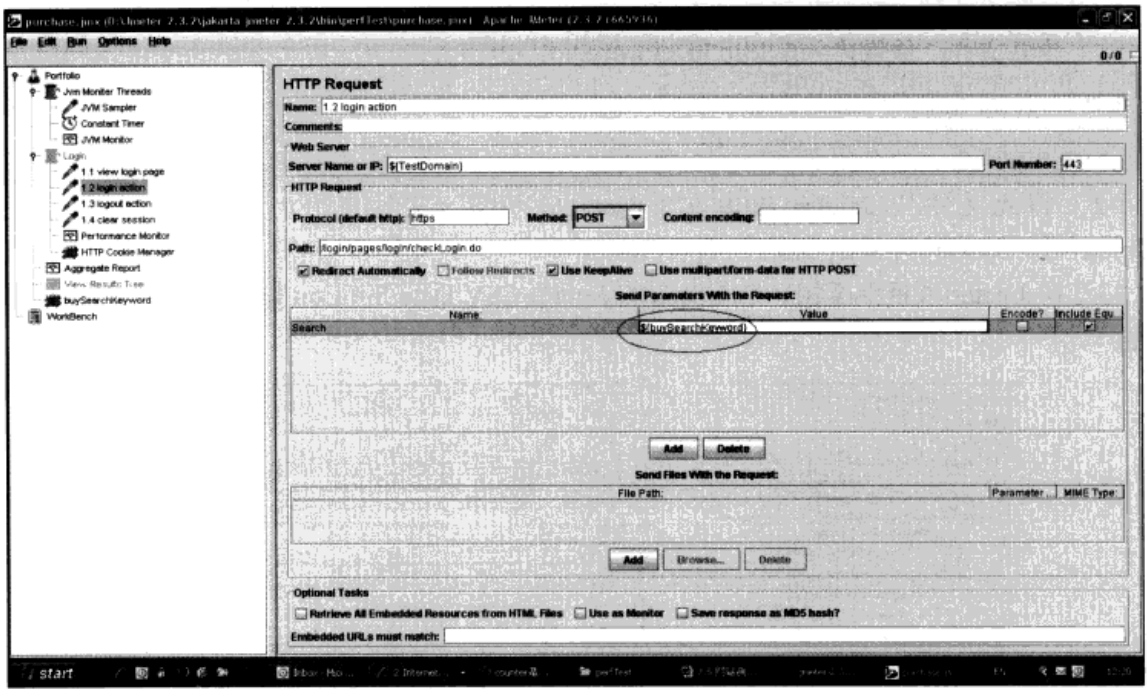


图 7-6 参数化替代

7.5.1.4 遇到的难点

在性能测试设计中会遇到一些难点。需要查阅网站数据，再进行分析确定。例如。

- 网站的并发用户数如何确定？
- 负载测试的时间如何确定？
- 思考时间（think time）如何确定？

另外，还有一些零零散散的问题需要查阅帮助文档或自己调查并试验，例如。

- 检查点的设定。
- 循环、远程控制等设计的实现。
- 集合点的设定。
- 自动发邮件功能的实现。

说明

我们使用了 Jmeter 自动发邮件通知用户的功能，这通过在 Listener 中选取 Mailer Visualizer 来实现。这需要下载 activation.jar 和 mail.jar，并将它们放在 lib 文件夹中。

- 如何在 Aggregate Report 中保存可以重新在 Jmeter 中打开的报告结果？

Offer!

说明

不能使用 Save Table Data，这会保存为.csv 文件，在 Jmeter 中打开会报错。我们只能在 Browse 按键左侧的 Filename 文本框中输入文件名（包括全路径），这样当 Jmeter 运行完毕后（即按下 Stop 或程序自然结束），会有一个.jtl 文件生成。该文件是可以在 Jmeter 中打开的，如图 7-7 和图 7-8 所示。

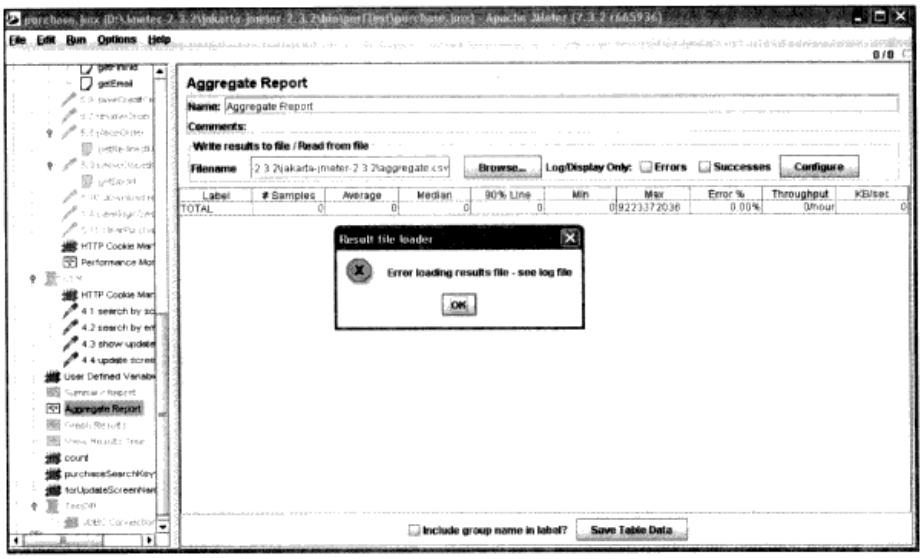


图 7-7 Jmeter 无法打开使用 Save Table Data 保存的. csv 文件

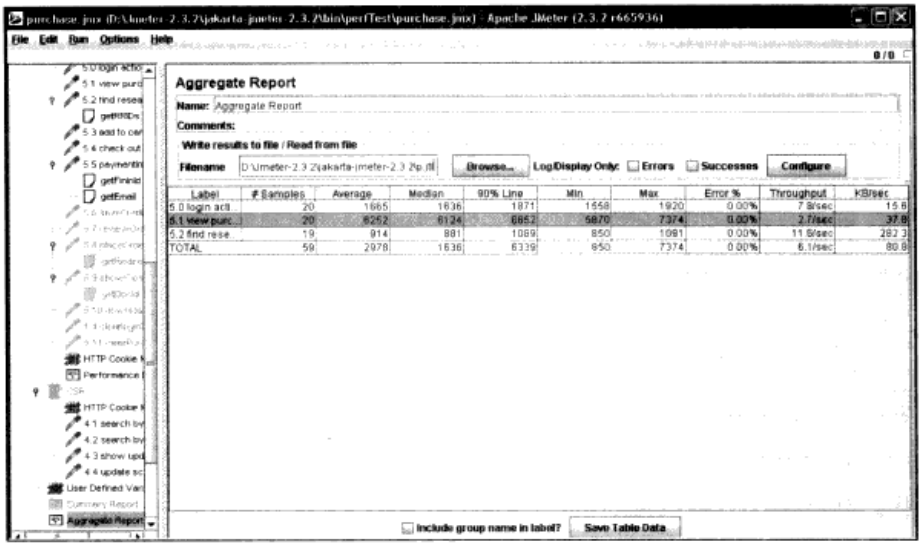


图 7-8 Jmeter 能正常打开 Filename 文本框中命名的文件

有趣的是，在 Filename 文本框中输入的文件名，即使使用.csv 甚至无后缀，也能被 Jmeter

打开。这说明文件名后被 Jmeter 自动添加了.jtl 后缀。

7.5.1.5 结果分析

在分析测试结果中，首先应当确立基线，即单线程时的测试结果（操作上可使用多线程取平均值的方法）。用此结果与并发等等测试结果对比。

CPU 占用、内存占用/变化、吞吐量、错误率及执行时间（主要指 90%line 时间，即 90%用户执行小于的时间）等是我们关心的主要指标。

例子

在大数据量测试中，我们观察多次垃圾收集前后的内存变化，如果垃圾收集后内存比测试开始时降低了不少，则说明内存泄露，如图 7-9 所示。

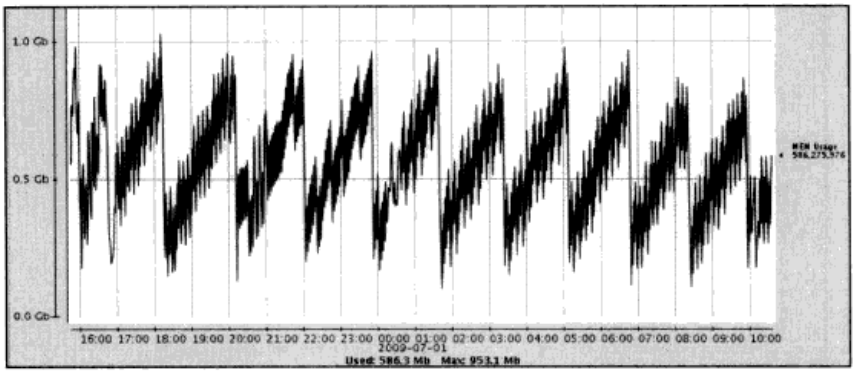


图 7-9 说明内存泄露

7.5.2 测试流程与计划

7.5.2.1 测试流程

具体测试如下。

（1）仔细研究需求和设计，判断风险所在。根据八二原则，在高风险处进行更多的精力投入，并估算测试时间、人员分配。

（2）对需求和设计不明确之处提出意见，与开发、设计人员讨论。

（3）紧扣需求写出测试用例。此时主要是通过性用例，用于测试系统的主要功能是否可以实现。

（4）设计一些攻击性测试用例，用来测试系统的鲁棒性。

（5）执行测试，同时补充测试用例。

（6）测试完毕后撰写测试报告。

（7）在完成的测试报告中继续补充测试用例，并等候下一轮测试开始。

（8）在新一轮测试开始时，首先验证提交缺陷的修改情况。再进行测试。若不止一个测试人员，最好将测试任务交换，以弥补单个测试工程师考虑不周及遗漏的情况。

Offer

（9）结束全部测试后记录每位测试工程师的绩效。

7.5.2.2 测试计划一例

欲测试一个 B-S 结构软件，有一个管理界面，通过查询数据库统计网站上销售各种财务报表的情况。对其分析并制订测试计划的要点如下。

（1）使用者仅限几个管理员，可确定不涉及性能问题，故不进行性能测试。

（2）通过因特网可以访问到该软件的 URL，故需要进行安全测试，此时主要是 SQL 注入（包括 URL 和登录界面两个方面）。

（3）管理界面的 GUI 测试需要做，但并非重点。

（4）由于涉及数据库查询，则需要对代码中的 SQL 语句进行白盒测试，这是重点。

（5）进行白盒测试后，还要确保这些 SQL 语句能够正常运行。因此要在界面上运行相应的功能，对比其查询的结果与数据库中查询的结果是否无误。

（6）由于有些软件缺陷不显示在界面上，而只显示在日志中，因此需要进行灰盒测试。打开日志，一边在界面上操作，一边观察日志，若出现未处理的异常被抛出，就记录下来，这是个软件缺陷。

（7）进行跨浏览器测试，主要是 Firefox、IE、Safari。

（8）Crontab（定时运行）是一个十分重要功能，由于它经常出问题，因此需要重点测试。

（9）对需求不明确之处提请设计及开发人员讨论。

（10）根据确定的测试范围，结合需求，撰写各部分测试用例。每部分用例都包括通过性和攻击性两部分。

（11）执行用例时过程中对用例加以补充。

（12）根据分析和项目时间一共计划了 3 轮测试。计划第 1 轮测试时间是 6 天，第 2、3 轮是 7 天。第 2、3 轮的第 1 天是返测所有状态为 Release、Reject、Reevaluate、Duplicate 的缺陷。测试的时间计划表如表 7-8 所示。

表 7-8　　测试的时间计划表

第 1 轮测试	第 1 天	界面及环境测试
	第 2～3 天	SQL 语句测试
	第 4 天	灰盒测试
	第 5 天	跨浏览器测试
	第 6 天	Crontab 测试
第 2、3 轮测试	第 1 天	返测 Bug
	第 2 天	界面及环境测试
	第 3～4 天	SQL 语句白盒测试
	第 5 天	灰盒测试
	第 6 天	跨浏览器测试
	第 7 天	Crontab 测试

Offer

7.5.3 测试策略举例

7.5.3.1 Crontab 简介

Crontab 的测试是使用 Linux 服务器的网站测试中的一个重要方面。Crontab 命令用于定时执行指定的命令，功能类似 Windows 下的“任务计划”，通常用来做一些重复性的工作。Linux 系统的用户只需将要执行的命令序列加到 Crontab 文件中，操作系统即会按用户配置的时间执行这些命令序列。

Crontab 文件中命令格式如下：

minute hour day-of-month month-of-year day-of-week commands

即：“分 时 日 月 周 命令”，共计 6 段，前 5 段都是时间，最后一段是要定期执行的指令。每段之间使用空格或制表符分隔。格式和解释如下：

```
# Use the hash sign to prefix a comment
# +---------------- minute (0 - 59)
#|  +------------- hour (0 - 23)
#|  |  +---------- day of month (1 - 31)
#|  |  |  +------- month (1 - 12)
#|  |  |  |  +---- day of week (0 - 7) (Sunday=0 or 7)
#|  |  |  |  |
#*  *  *  *  *  command to be executed
```

/var/adm/cron/cron.allow 和 /var/adm/cron/cron.deny 文件控制哪一个用户可以使用 Crontab 命令。root 用户可以创建、编辑或删除这些文件。这些文件的条目是用户登录名称，每个名称占一行。如果登录标识和多个的登录名称相关联，则这个 Crontab 命令采用第一个在 /etc/passwd 文件中的登录名称，而不管当前实际使用哪一个登录名称。另外，若要允许用户启动 cron 作业，则应该使用 chuser 命令将 /etc/security/user 文件中的守护进程属性设置为 True。

7.5.3.2 测试方法

这里我们就来谈一下 Crontab 的测试方法。这种测试方法是灰盒方法，需要查看数据库、软件程序，另外还要在界面上进行操作。下面以一个 B-S 结构的软件为例，该软件使用了应用服务器来连接数据库服务器。网站主要使用 JSP 进行编写。

请看一个普通的 Crontab 程序（里面包含了多个定时运行的程序）：

```
15 18 * * * /bin/bash -l /rt/siqi/bin/siqi_renewal.sh
0 23 * * * /bin/bash -l /rt/siqi/bin/siqi_billing.sh
5 6 * * * /bin/bash -l /rt/siqi/bin/siqi_abandonedcart.sh
14 8 * * * /bin/bash -l /rt/siqi/bin/siqi-annualsubnotif.sh
0 19 * * * /bin/bash -l /rt/siqi/bin/siqi-japanesesubnotif.sh
```

现在我们选择 1 个 Crontab 脚本程序 siqi_billing. sh 来测试。该脚本程序的目的是每天扫描所有购买财务报告的用户，是否退订。如果没有退订，并且超过了 1 年，则自动续订并收费。

测试步骤如下。

（1）查看 shell 程序源代码，看看调用了哪个类。查看类的 Java 代码，分析类中的方法。

Offer

例如，siqi_billing.sh 源码如下：

```
CP="$CLASSPATH:/rt/siqi/conf/jpa"

CRONS_HOME=/rt/siqi
for i in `/bin/ls "$CRONS_HOME"/lib`
do
  CP="$CRONS_HOME/lib/""$i":"$CP"
Done

KEYS="  -Drcommerce.certs.dir=${CRONS_HOME}/conf/certs  -Drcommerce.key.file.cookie=${CRONS_HOME}/
crypto/key -Drcommerce.key.file.password=${CRONS_HOME}/crypto
/key  -Drcommerce.key.file.creditcard=${CRONS_HOME}/crypto/key  -Dcrons.home=${CRONS_HOME}  -Dlog4j.
configuration=file:${CRONS_HOME}/conf/log4j_billing.xml –Dap
plicationContextFile=applicationContext.xml"

"$JAVA_HOME"/bin/java $DEBUG_OPTS ${KEYS} -cp "$CP" com.siqi.ecommerce.finance.BillingCron >"$CRONS_
HOME/logs/BillingCron.log" 2>&1
```

从脚本代码中，明显地，可以发现 1 个类，com.siqi.ecommerce.finance.BillingCron，被使用。再分析这个类的 Java 代码,发现 RenewalTask.processOneReport (oneReport)方法被调用。

（2）查看方法源代码，并查看调用了数据库中的哪些表和哪些字段，对这些表和这些字段有些什么操作。

（3）根据需求制订测试用例，测试用例包括通过性和攻击性两部分。

（4）执行测试。

（5）提交测试结果。

（6）追踪缺陷的解决情况。

7.5.3.3 测试用例

根据分析写出部分测试用例，如表 7-9 所示。

表 7-9 部分测试用例

用　例	测试步骤	预期结果
1	1．建立新用户 2．购买 1 年期财务报告 3．在数据库中修改购买时间为 1 年零 1 天前 4．立即运行 siqi_billing.sh 5．查看结果（包括界面上和数据中）	续订成功，数据库的数据会做相应更新
2	1．建立新用户 2．购买 1 年期财务报告 3．在数据库中修改购买时间为 1 年差 1 天前 4．立即运行 siqi_billing.sh 5．查看结果（包括界面上和数据中）	续订没发生



续表

用　例	测试步骤	预期结果
3	1．建立新用户 2．购买1年期财务报告 3．在数据库中修改购买时间为3年前 4．立即运行 siqi_billing.sh 5．查看结果（包括界面上和数据中）	续订成功，只收费1次而不是3次，数据库的数据会做相应更新
4	1．建立新用户 2．购买1年期财务报告 3．在数据库中修改购买时间为1年零1天前 4．拔掉网线 5．立即运行 siqi_billing.sh 6．查看结果（包括界面上和数据中）	续订不成功，重试3次（注：是否重试只和该脚本有关，不通用，下同）后，系统发出邮件通知管理员。错误日志记录了这些信息
5	1．建立新用户 2．购买1年期财务报告 3．在数据库中修改购买时间为1年零1天前 4．拔掉网线 5．立即运行 siqi_billing.sh 6．在应用服务器试图连接数据库的重试时间内，插上网线。 7．查看结果（包括界面上和数据中）	续订成功，数据库的数据会做相应更新。重试情况记录在日志中
6	1．建立新用户 2．购买1年期财务报告 3．在数据库中修改购买时间为1年零1天前 4．停止数据库运行 5．立即运行 siqi_billing.sh 6．在应用服务器试图连接数据库的重试时间过后，启动数据库。 7．查看结果（包括界面上和数据中）	续订不成功，重试3次后，系统发出邮件通知管理员。错误日志记录了这些信息
7	1．建立新用户 2．购买1年期财务报告 3．在数据库中修改购买时间为1年零1天前 4．停止数据库运行 5．立即运行 siqi_billing.sh 6．在应用服务器试图连接数据库的重试时间内，启动数据库。 7．查看结果（包括界面上和数据中）	续订成功，数据库的数据会做相应更新。重试情况记录在日志中
8	1．建立新用户 2．购买1年期财务报告 3．立即退订 4．在数据库中修改购买时间为1年零1天前 5．立即运行 siqi_billing.sh 6．查看结果（包括界面上和数据中）	续订没发生

Offer

引申

可能有读者注意到这里没有退订和续费同时发生的测试用例。因为第一，我们无法做到测试时完全同时；第二，如果被收费，这个时候收费是有道理的，不会有用户细究这个细枝末节，用户会从自身找原因。

7.5.3.4 可能的软件缺陷

- Crontab 不执行（原因往往是因为环境变量不正确）。
- Cron job 的运行结果与需求不符。
- 未能发出邮件。
- 程序中对数据库操作有误。
- Crontab 文件格式写错。
- Shell 脚本调用错了 JSP 的类。
- JSP 代码错误。
- 混淆了服务器时间与本地时间。
- 错误处理机制不健全。
- 第一个 Cron job 尚未完成，第二个已经开始，次次如此。
- Cron job 日志中未处理异常及错误（错误不一定输出，这要看用户指定输出的级别 Error、Warning、Info 等）。

Offer



第五部分

测试英文篇

在第四部分测试实战篇中，我们实际应用了测试理论篇和测试开发篇的内容。通过对测试项目的分析，我们举例说明了制订测试计划、设计测试用例、开发测试工具、绩效评定、测试网络、自动化测试脚本与工具应用等实战经验。我们也给出了一些典型面试题，用以使大家举一反三。

在本篇中，我们谈一下软件测试中的英文运用。众所周知，IT 外企的笔试题是全英文的；在面试和电话面试的对答中英文也是占大部分的；面试中还经常会要求应聘者用英文写一些内容。可以说英文能力的强弱直接影响到面试结果，毫无疑问也将影响到薪水的高低。在进入 IT 外企后，英文水平的高低直接影响到能否升迁，一般情况下，一个不能自由运用英文的员工，是不可能成为中管或高管的。

例子

记得在汤森路透入职后的第一堂培训课上，老师就指出了汤森路透的工作中最重要的 3 点。其中第 3 点就是要求员工全力提高英文能力。

说明

我这里提到的自由运用是指。

- 听：从听说中文状态随时可以转化为听英文状态，在有人同时说中文的环境中仍不受影响，能顺利听懂一般的方言比如印度英语，对于新闻、数字、演讲等，同步完全听懂。
- 说：从听说中文状态随时可以立即转化成说英文状态，流利自然，基本没有语法错误。
- 读：达到读中文的速度，基本不用查字典。
- 写：至少达到“信达雅”的“信”，基本准确地表达意思。

译：能用中文和英文互相翻译正确细节即可。至于“雅”，超越了一般的自由运用水平，我们不做要求。

强调

不是一定要出国才能练好英文，在国内把英文练过关的人比比皆是。关键还是能否下苦功以及能否坚持。

当然，对于普通的测试工程师而言，应付一些基本的工作，大学英语四级水平就勉强达标了。

下面，请大家跟随我进入测试英文的领域。

第8章 典型英文面试笔试题

8.1 如何提高英文水平

关于如何提高英文，很多前辈都有专门论述。我们几位作者水平有限，只是从自身的经验出发，提出一些自己的看法。

我们认为在听说读写译等各种能力中，听力最重要。听是说的基础，只有当你理解对方意思时，才可能进行适当的反应，因此我们应当把听力作为最重要的能力来进行训练。我们经常发现，在外企中，不少人说起来还算流利，可是听力不过关。这大大阻碍了他们的发展。这样的水平，实际上在英文上并没有突破真正的关卡，没有进入“自由王国”。英语真正过关的标志关键是在于听力过关。

听、说、读、写相辅相成，尽量将中英文统一。译自成一体，寻找中英文的不同点。

背诵是英文的基础。范文、词汇、短语、句型都需要大量的背诵。

说和写有相通之处，它们都是首先要在脑中组织好句子。我们主张口语的语法要正确，反对仅表达出意思却不注重语法的低标准。

听力和词汇量是英语进步的阶梯。如果想进一步成为高管，这两个瓶颈一定要突破。

做题是很好的训练方法。听读写译这 4 种能力都可以通过做题来提高。至于说，一是要背诵大量的句型、文章、词汇、短语，二是要创造机会去说。每天自己也可以给自己一个主题，自己对自己说。

下边对听说读写译这 5 种能力的培养加以论述。其中有的论述已经超越了软件测试工程师这个层次，实际已经是作者关于英文之“道”的一些思考。当然这些不是无用功。我们相信，读完本节，大家将对软件测试工程师如何提高英文这一问题有着更高层次的理解。

8.1.1 通过听收音机和做听力题来练习听力

通过看电视或看电影来练习听力，笔者个人认为不是一个很好的方法。因为有字幕时，往往我们会去读字幕，往往是读懂而非听懂。在没有字幕时，剧情和背景也会帮助我们猜出说话内容。



在看电视或电影时，建议可以关掉屏幕，单纯来听，实际上这样和听收音机没有什么区别。

我们提倡使用收音机来训练听力。这是因为，收听收音机避免了我们用看或是用猜的方式判断谈话内容，我们只能靠听觉，无法投机取巧。CRI、BBC、VOA 都是很不错的练习听力的电台。

我们在听收音机时，不能在精神不振的时候听（例如睡前），应当在注意力集中的时候来听。同时准备好纸笔，遇到听不太懂的地方记下发音，之后再查找这个词。这样一来，下次再遇到这个词，就能够听懂。

同时我们要掌握一些常用的口语句型。建议大家阅读一些雅思和托福考试的听力书，背诵一些句型。

另外，我们还可以做听力题。例如历年的托福、雅思及四六级真题都是很好的训练题目。大家可以去寄托天下网站（www.gter.net）下载听力题，当年笔者考 GRE 和托福时经常上这个网站，这个网站上有大量的免费真题。

做题后，不要急着看答案。凡是没听懂的地方，应反复听直到听懂，然后再对照答案。

真正到使用听力的时候，基本上只可能说一遍，因此平时的训练就是尽量要求自己一遍就听懂。

可以做一段时间听力题，听一段时间收音机，再做听力题，再听收音机。以做题来检验自己的听力水平是否有提高。

另外，不要只听标准美音和英音。如同在中国存在大量的方言一样，在英语世界里，也存在大量的方言，如印度和巴基斯坦腔调、苏格兰口音，都不易听懂。我们学英语是为了实用不是花哨，我们会经常遇到有口音的老外（例如笔者目前的领导就是两个印度人），因此我们要努力练习听懂口音。实际上，如果练到连口音都听懂了，听标准音更是不在话下。

8.1.2 通过背诵、聊天、写作来练习口语

不论是希望出口成章，还是希望说得地道，背诵都必不可少。

背诵范文，不但可以帮助我们提高写作水平，同时也能促进我们的口语水平。因为我们在说之前，通常都会在脑中组织一下语言，这种组织的语言最后落在纸上就是写作，若最后说出来就是口语。

常用的、地道的句型需要背诵。目前这类书籍市面上有很多，如《英语 900 句》系列等。

聊天是实战，当然对提高口语非常有帮助。

如果大家找不到外国朋友来练习口语，完全可以去各高校的英语角。一般来说，各高校基本每周都有英语角的活动，大家在那里可以练习口语，有时能碰见外国朋友。

⚠ 警示

当找练习口语听力的对象时，找中国人练习口语还行，练习听力不成，因为对方的发言方式和腔调你比较熟悉。



强调

大家都要尽量注意语法的正确，不要一出口就满篇语法错误，尽管意思表达出来了，但这样你的口语水平不会得到实质性的提高，建议大家都以高标准要求自己。

8.1.3 通过背单词、看报纸来练习阅读

根据笔者的经验，即使考过大学英语 6 级，看英文报纸仍是比较吃力的。原因无它，就是词汇量不足。

要背多少词汇才够？

当年笔者在备考 GRE 初期，当背到 12000 个词汇量时，看报纸就没有那么吃力了。因此，12000 个词汇量大概是看报纸最初步的要求。

怎样背单词效果好？

我建议大家首先从托福、雅思词汇开始背。

词汇书中的单词通常是按字母顺序排列的。通常在页面中，左边是英文，右边是中文释义。请按照以下方法背 8 遍，如表 8-1 所示。

表 8-1 背诵单词书的方法

第 1 遍	A-Z，顺序，盖住右边的情况下能记住右边释义
第 2 遍	Z-A，逆序，盖住右边的情况下能记住右边释义
第 3 遍	打乱顺序，盖住右边的情况下能记住右边释义
第 4 遍	打乱顺序（与第 3 遍不同顺序），在盖住右边的情况下能记住右边释义
第 5 遍	A-Z，顺序，在盖住右边的情况下能记住右边释义，在盖住左边的情况下能记住左边英文
第 6 遍	Z-A，逆序，在盖住右边的情况下能记住右边释义，在盖住左边的情况下能记住左边英文
第 7 遍	打乱顺序，在盖住右边的情况下能记住右边释义，在盖住左边的情况下能记住左边英文
第 8 遍	打乱顺序（与第 7 遍顺序不同），在盖住右边的情况下能记住右边释义，在盖住左边的情况下能记住左边英文

经常请大家考考你背诵，或者自己考考自己。每天都要背，不能停。每天背诵之前应当先复习一下昨天背诵的单词。每周一应复习上周背诵的单词。每个月初应复习上个月背诵的单词。

自己可以制作些卡片，每天可以更换不同的卡片。

背单词要多利用“边边角角”的时间。排队时、坐车时、睡前、早起等都可以是背单词的时间。

背单词是辛苦活，以每天只背 100 个计算，100 天（3 个多月）才能将 10000 个单词背 1 遍。可是显然 1 遍是肯定记不住的，所以要严格要求自己，每天要尽量多背一些。记得在考 GRE 前的 3 个月，每天我最少要背 800 个单词。

可能你看到一些人写的议论 GRE 的文章，谈到背单词时是多么郁闷。根据笔者个人的体会，与笔者在英国艰苦卓绝的读研生活比起来，备考 GRE 就像休息一样。所以我可以负责任地告诉你，



你对背单词坚持不下来的原因并不是多苦多累而是你软弱了，因为你没有经历过更苦更累的学习。

引申

很多人都抱怨有的GRE单词一辈子都不一定用得到。实际上，除了个别的，大部分GRE单词并不一定罕见。笔者认为这些词汇大都有用。有的人怕苦怕累所以抱怨，青年人要勇往直前，不要被这些怕苦怕累的思想拖后腿。

看报纸对词汇量是个很好的检验。刚开始一定是非常吃力的。后来慢慢就看得快起来了。再后来，你不认识也能猜出来单词大概的意思。最后，一张报纸上找不出来几个你不认识的词汇。

思索

词汇量是可以在短时间内大幅度提高的。进一步地，对每一本词汇书而言，笔者认为背单词的时间不宜拖得太长，否则背了后面忘了前面，效果不佳。

当你词汇量越来越多的时候，背新单词也会越来越容易。英文会越来越融入你的血液。

8.1.4 通过背诵和请高手修改来练习写作

熟读唐诗三百首，不会吟诗也会吟。

背诵范文、词汇、句型、名言、典故，在英文写作中也同样非常重要。

引申

有人可能会问那些唐诗作者是怎么写出的诗？他们是否不用背诵？答案是他们也要大量地背诵更古老年代的经典文学，例如杜甫就从《楚辞》中汲取过很多营养。所以背诵甚至对任何一个作家都非常重要，更不用说对我们普通人的写作。

在写作中，需要注意文章间架结构以及炼字等推敲，这些能力和大量的背诵是分不开的。

写作后请高手修改非常重要。写完一篇文字，如同开发人员完成一个项目，自己往往找不出来错误，需要测试人员来帮助寻找Dug。在高手看来，我们的文字必然有很多需要修订之处。高手的指点会带来新视点、新角度，犹如一汪清泉和一盏明灯，使你豁然开朗。

例子

记得大学时在校刊上发表文章后，每次都分析那些被编辑老师修改过的地方。为什么老师这么改？仔细思索之后，往往会有笔力上的提高。

对于测试工程师而言，有时要写比较复杂的文档或邮件。当你写完后，最好请英文水平高的同事修改一下，然后你再仔细体会一下别人的修改，肯定会有收获。

8.1.5 通过范例与做题来练习翻译

其实作为一个测试工程师，我们不需要太精深的翻译水平，能翻译出大概正确的意思，达到“信达雅”中的“信”就足够了。不过既然谈到此处，笔者也谈一点对翻译的认识。大家权当放松。

笔者曾幸运地得到过商务印书馆的英文编辑的笔试。在英文方面下苦功多年，笔者自认听说读写的水平还说得过去，没想到，就栽在这个“译”上了。

考过研的学生肯定比那时的笔者清醒。在笔者参加的托福、GRE 等一系列英文考试中，没有考查过“译”这种能力。自小到大，所有的英文老师都告诉笔者，要以英文来理解中文，因此笔者从未专门训练过“译”这种能力。在当时的笔者看来，自胡适、陈独秀等人主张白话文运动以来，中英文近似一样，连语序都差不多，而且，笔者一直也喜欢文学和历史，都发表过一些中文文章，因此笔者错误地以为译的能力不错，实则大谬不然。在专业译者来看，中英文差距甚多。

笔者粗浅地认为，译是在于寻找中英文的不同点。在应聘商务印书馆失败后，笔者研究了一些专业翻译理论书籍，同时也找来一些考研的翻译题加以训练。慢慢地，笔者也寻找到中英文的一些不同之处，后来也在杂志上发表过一些翻译文章。在这里，笔者向大家推荐陈文伯编著的《译艺》。

思索

翻译既然是寻找中英文的不同点，是否会伤害英文的感觉呢？还是说大成之后，中英文浑然一体，而此过程中译的能力能帮助我们达到大成？限于中文和英文的水平，这点我一直没有想通。诸君若有悟道者，请以教我，不胜感激。

练习翻译能力，理论学习非常重要。在日常的练习中，我们可以找到名家翻译的作品，先试译一下，再细细研究原文。对低层次的非专业人士来说，考研的翻译真题也是很好的练习材料。

8.2 软件测试面试英文题必知必会基础

软件测试概念和原理是软件测试工程师面试中考查的重点。

扎实的理论基础可以指导实践。知其然不知其所以然是不可取的。仅凭经验工作而不具备理论基础，通常会犯只见树木不见森林的错误，缺乏全局观系统观。一旦深入思索问题，没有成套理论体系的工程师，很容易迷失方向。

因此，掌握系统的软件测试概念与原理知识，毫无疑问是一位优秀的软件测试工程师的基础技能。

其实英文或中文只是包装不同，内在的知识是相同的。本节涉及的知识在第 3 章中已进行过系统讲解，此处仅仅是换用英文来提问和回答。

本节选取了 5 道典型题目，对提问和回答给出了中英文互译，并对涉及专业知识加以分析。

面试试题 1

（1）What does 80-20 principle mean in software testing?

（2）What is the meaning of smoking test?



译文

（1）软件测试中的八二原则是什么？
（2）冒烟测试是什么？

解答

（1）80 percent errors happen in 20 percent part of the tested software,80 percent test resources should be applied for 20 percent, high-risk modules.
（2）Smoking test is one kind of acceptance test. Once a software version passes smoking test, the next test phase will begin.

解答译文

（1）80%的错误发生在20%的地方，80%的测试安排在20%高风险的模块。
（2）冒烟测试是一种验收测试。当一个软件版本通过验收测试，即意味着下个测试阶段可以开始。

分析

- 此处高风险指的不是错误最多的模块，而是指失效可能性、失效影响值相加后得到的优先级高的模块。
- 在冒烟测试中，依照明确的测试用例，测试人员将少数关键功能在新的软件版本上进行测试，只有通过了这种测试，该新软件版本才被测试部门正式接受，之后该软件版本会进入全面测试。如果没有通过冒烟测试，则测试部门会拒绝这个版本，然后版本将会被退回开发与集成部门，直到解决了这些问题为止。

面试试题2

In spiral model of software engineering, in which phase should software test begin?

译文

在软件工程的螺旋模型中，软件测试应当在哪个阶段开始?

解答

The phase of drawing up user requirements.

解答译文

在用户需求阶段。

分析

- Spiral model，螺旋模型。1988年，BarryBoehm正式发表了软件系统开发的“螺旋模型”，它将瀑布模型和快速原型模型结合起来，强调了其他模型所忽视的风险分析，特别适合于大型复杂的系统。螺旋模型是一种演化软件开发过程模型，它兼顾了快速原型的迭代

Offer

的特征以及瀑布模型的系统化与严格监控，它最大的特点在于引入了其他模型不具备的风险分析，使软件在无法排除重大风险时有机会停止，以减小损失。同时，在每个迭代阶段构建原型是螺旋模型用以减小风险的途径。

螺旋模型沿着螺线进行若干次迭代，图 8-1 中的 4 个象限代表制定计划、风险分析、实施工程、客户评估。

螺旋模型由风险驱动，强调可选方案和约束条件从而支持软件的重用。在任一阶段，首先确定阶段目标、选择方案、约束条件，之后对开发策略进行风险分析。若一些严重的风险无法被排除，则该方案需要立即被终止，否则将启动下个开发步骤。最后，评价该阶段结果，并设计下个阶段，如图 8-1 所示。

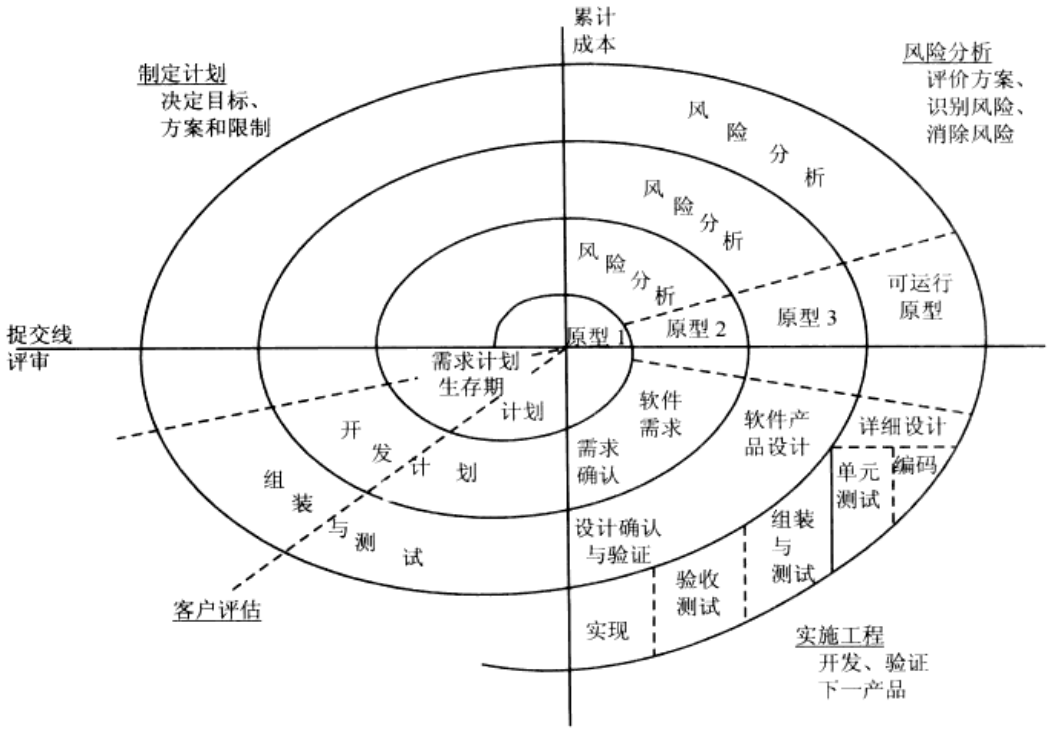


图 8-1 螺旋模型

- In … phase，意为：在……阶段。
- drawing up，意为：制订。
- user requirements，意为：用户需求。
- 本题若回答“设计阶段”，不得分。因为测试从需求开始是软件测试的基础认识之一。

面试试题 3

What are the relations and difference among performance test, load test and stress test?

译文

性能测试、负载测试及压力测试的区别和联系是什么?

解答

Performance test includes the other 2.

Stress test: the test of seeking for performance extrema.

Load test: the test of examining whether system performance reaches requirement goal.

解答译文

性能测试包括后面二者。压力测试是未知性能临界点而寻找临界点的测试。负载测试是测试系统性能是否达到需求的测试。

分析

- 这是一道非常常见的面试题，主要考查压力与负载测试的概念。

面试试题4

What is the difference between regression test and retest/verification?

译文

回归测试和返测有何区别?

解答

Regression test works after some code change to find new imported bugs(regression bugs) as well as to check whether found bugs are fixed. All or part of previous test cases will be executed according to the influence of code change in regression test. Regression test survives in system test, integration test and unit test.

Retest or verification means to test whether a specific bug is fixed.

解答译文

回归测试通常在修改一定量的代码后进行，目的是检测出由于代码修改而引入的新缺陷，即回归缺陷，还有就是检测已改正的软件缺陷是否仍旧存在。当回归测试进行时，将根据代码修改影响的范围，重复以前的全部或部分测试用例。回归测试在系统测试、集成测试和单元测试阶段都会进行。

返测是检测某个特定 Dug 是否已被解决的测试。

分析

- 回归测试是非常重要的。由于代码之间的逻辑关联，理论上，每修改 3~5 个软件缺陷，就会引入 1 个新的缺陷。

Offer

例子

2008年秋，当时笔者在***公司工作。由于发现软件缺陷数量较多，曾有机会与手机测试部门最高负责人交流。他告诉我，在手机部门，平均每修改 10 个缺陷就会引入 3 个新缺陷。

- 通常在缺陷跟踪系统中，当开发人员修改了一个 Dug，并且包含修改的新软件版本正式发布后，开发人员会将此 Dug 的状态标记为类似“Resolved”或“Release to test”，测试人员会收到缺陷跟踪系统的来信告知这一点。于是测试人员会对这个 Dug 进行返测，如果这个 Dug 已被改正，则测试人员会将其状态改为“Closed”。

面试试题 5

In White-box test methods。
（1）Could you explain what Condition Coverage is?
（2）What is meaning of Decision Coverage?

译文

在白盒测试方法中。
（1）条件覆盖是什么？
（2）判定覆盖是什么？

解答

（1）Condition Coverage: Has each boolean sub-expression evaluated both to true and false?
（2）Decision Coverage: Has each control structure (such as an IF statement) evaluated both to true and false?

解答译文

（1）条件覆盖：设计足够多的测试用例，运行所测程序，使程序中每个判断的每个条件的每个可能至少执行一次。
（2）判定覆盖（分支覆盖）：设计足够多的测试用例，运行所测程序，使程序中每个判断的取真分支和取假分支至少各执行一次。

分析

- 本题译文是意译，不是直译。在学习翻译的过程中，我发现主流的翻译思想是提倡意译而非直译。
- 条件覆盖和判定覆盖都属于代码覆盖，它们是白盒测试中的两种测试方法。代码覆盖是白盒测试的一个分支，它还包括。

Offer

（1）语句覆盖（Statement Coverage）。

设计足够多的测试用例使每行代码至少被执行一次。

（2）修正的条件/判定覆盖（Modified Condition/Decision Coverage (MC/DC)）。

这个覆盖度量标准需要足够的测试用例来确定各个条件能够影响到包含的判定的结果。它要求满足以下两个条件。

a. 每一个程序模块的入口和出口都要考虑至少被调用一次，每个程序的判定到所有可能的结果至少转换一次。

b. 程序的判定被分解为通过逻辑操作符连接的布尔条件，每个条件对于判定的结果值都是独立的。

（3）路径覆盖（Path Coverage）。

通过设计足够多的测试用例，使得运行这些测试用例时，程序的每条可能执行到的路径都至少被经过一次（如果程序中有环路，则要求每条环路至少经过一次）。

（4）判定/条件覆盖（Decision/Condition Coverage）。

通过设计足够多的测试用例，使得运行这些测试用例时，判定中的每个条件的所有可能结果至少出现一次，并且每个判定本身的所有可能结果也至少出现一次。

（5）条件组合覆盖（Multiple Condition Coverage，MCC）。

通过设计足够多的测试用例，使得运行这些测试用例时，每个判定中条件结果的所有可能组合至少出现一次。

- 除了以上这些，代码覆盖中还包括函数覆盖（Function Coverage）、调用覆盖（Call Coverage）、线性代码顺序及跳转覆盖（Linear Code Sequence and Jump Coverage）、数据流覆盖（Data Flow Coverage）、目标代码分支覆盖（Object Code Branch Coverage）、循环覆盖（Loop Coverage）、竞争覆盖（Race Coverage）、关系操作符覆盖（Relational Operator Coverage）、弱变化覆盖（Weak Mutation Coverage）、表覆盖（Table Coverage）、跳转间路径覆盖（JJ-Path Coverage）、函数入口/出口覆盖（Entry/Exit Coverage）等。

8.3 软件测试面试英文基础——计算机应用

计算机应用是软件测试工程师面试中考查的另一个重点。

软件测试的基本概念和原理是基于计算机应用基础之上的，也就是说，没有计算机应用的基础，空谈测试方法理论，不过是空中格阁。软件测试基本概念和原理掌握得再好，如果没有强大的计算机应用基础，面试失败的可能性仍然非常大。

请大家牢记，软件测试工程师首先是软件工程师，如果连计算机应用的基础知识都没有掌握，怎么能算是合格的软件工程师？

毫无疑问，广泛深厚的计算机应用知识，是成为一流的软件测试工程师的必备技能。



本节涉及的知识在本书第 4 章中已有过系统讲解，此处仅仅是换用英文来提问和回答。

本节选取了 5 道典型题目，涉及性能测试、数据库、配置管理、质量管理和网络知识这 5 个领域。笔者对提问和回答给出了中英文互译，并对涉及专业知识加以分析。在本书第 4 章中包括了本节中涉及的 5 个领域，我们分别论述了计算机应用的 16 个领域。本节的 5 道题仅仅只是引导大家以管窥豹，更全面的计算机应用基础知识和更多领域的知识需要大家花费心血去努力掌握。

面试试题 1

（1）What is IP Spoofing in Loadrunner?

（2）Which script language is used in Loadrunner programme?

（3）Can cycle statement be used in the script language?

译文

（1）LoadRunner 中的 IP 欺骗是指什么？

（2）LoadRunner 的脚本编程是用什么语言？

（3）能否使用循环语句？

解答

（1）In performance test, the situation of many concurrent requests against a server from different hosts is always tested. To simulate this, “IP spoofing” is used in Loadrunner. “IP spoofing” enables each host machine to “spoof” the server into thinking it is using many different IP addresses.

（2）Java, C, Visual Basic and VBScript are supported by Loadrunner. C is default script language.

（3）Yes.

解答译文

（1）在性能测试时，经常需要检测大量主机访问服务器的情况。为了仿真这一点，LoadRunner 使用了 IP 欺骗，即一台客户机虚拟出多个 IP 地址，用于“欺骗”服务器，使服务器误以为是不同客户机发出的请求。

（2）LoadRunner 脚本支持的语言有：C、Java、Visual Basic、VBScript，默认的脚本生成语言为 C。

（3）可以使用循环。

分析

- 此处高风险指的不是错误最多的模块，而是指失效可能性、失效影响值相加后得到的优先级高的模块。



- 在冒烟测试中，依照明确的测试用例，测试人员将少数关键功能在新的软件版本上进行测试，只有通过了这种测试，该新软件版本才被测试部门正式接受，之后该软件版本进入全面测试。如果没有通过冒烟测试，则测试部门会拒绝这个版本。该版本将会被退回开发与集成部门，直到解决了这些问题为止。
- 设置 IP 欺骗的原因：
 - 当某个 IP 的访问过于频繁或访问量过大时，服务器会拒绝访问请求。
 - 有些服务器配置了负载均衡，使用同一个 IP 不能测出系统的实际性能。
- 使用 IP 欺骗时要注意：
 - 虚拟 IP 是同一个 Generator 上的多个 IP，这种分配过程由 Controller 自动来进行。
 - 对于同一个 Generator，只需要以其中某个 IP 添加到 Generator 中即可，用不同 IP 添加多次是没有必要的。
 - 对于同一个 Generator，在模拟用户数量多于 IP 数量时会发生 IP 重复的现象，否则将随机分配不同的 IP。
- LoadRunner 使用的脚本语言：
 - 对于 FTP、COM/DCOM 和邮件协议（IMAP、POP3 和 SMTP），VuGen 还可以使用 Visual Basic、VBScript 和 JavaScript 来生成脚本。
 - C 语言：以复杂的 COM 构造、C++对象来录制应用程序时使用 C 语言。HTTP 协议的脚本也默认使用 C 语言。
 - Visual Basic：用于基于 Visual Basic 的应用程序。
 - VBScript：用于基于 VBScript 的应用程序，如 ASP。
 - JavaScript：用于基于 JavaScript 的应用程序，如 js 文件和动态 HTML 应用程序。

面试试题 2

Which is incorrect in these 4 opinions for relational database?

A. Components in each column are the same kind from the same domain.

B. Dada in different columns are from the same domain.

C. The order of any 2 rows can be exchanged, but columns can not.

D. Any 2 tuples should not be the same in a relation.

译文

对关系数据库而言，下面说法错误的是。

A. 每一列的分量是同一种类型数据，来自同一个域。

B. 不同列的数据可以出自同一个域。

C. 行的顺序可以任意交换，但列的顺序不能任意交换。

D. 关系中的任意两个元组不能完全相同。

解答

答案是 C。

分析

• 关系数据库对关系有如下的限定。

（1）属性不可分解。

（2）关系模式中属性的数据类型及属性的个数是固定的，并且每个属性必须命名。在同一个关系模式中，属性名必须是不同的。

（3）一个关系仅仅有一种记录类型，即一种关系模式。

（4）在关系中元组的顺序（行序）是无关紧要的。

（5）在关系中属性的顺序（列序）可以交换，但交换时应连同属性名一起交换；否则，必须保持这个顺序。

（6）同一个关系中不允许出现完全相同的元组。

关系表中的一行称为元组，元组可表示一个实体或实体之间的联系。

面试试题 3

What are the deficiencies of VSS, a configuration management tool, against CVS?

译文

配置管理工具 VSS 相比于 CVS 的缺点是什么？

解答

（1）We need purchase to buy VSS made of Microsoft. While the open source project, CVS, is free.

（2）VSS only supports Microsoft Windows platform without cross-platform ability.

（3）VSS does not support version branch and merge, developers need manually create and maintain version trees. Change in one version tree has to be processed again in other version trees, this results in duplicate efforts and many errors.

（4）VSS does not provide cooperative communication mechanism. It does not support automatic building and development flow.

（5）VSS does not support parallel development, only one person can modify one file at a moment.

（6）Poor elasticity of VSS makes it to not suit for a project containing over 10 developers in practice.

（7）VSS does not support distributed development, developers cannot stay in different cities.

（8）Malfunctions always happen in version warehouse owing to poor stability of VSS.

Offer

Actually suite tools of Microsoft may not restore VSS database.

（9）VSS is difficult to cooperate with defect tracking systems to perform ideal configuration management because of its weak customization ability.

解答译文

（1）VSS 是微软产品，需要付费购买，CVS 是开源的，不需付费。

（2）VSS 只能支持 Microsoft Windows 平台，不具备跨平台能力。

（3）VSS 不支持版本分支和合并，开发者需要手工创建维护多个版本树，在一个版本树上进行的修改必须在其他版本树上再进行一次，这样将导致重复劳动和出现大量错误。

（4）VSS 不提供辅助通信机制，不提供自动构建支持，不支持开发流程化。

（5）VSS 不支持并行开发，在一个时间里能有一个人修改某个文件。

（6）VSS 伸缩性差，在实践中不适合超过 10 个开发人员项目。

（7）VSS 不支持分布式开发，开发人员不能位于不同城市。

（8）VSS 不稳定，其版本仓库时常发生故障，且微软的配套修复工具未必能修复 VSS 数据库。

（9）VSS 定制能力弱，很难与缺陷跟踪系统集成为完整的配置管理解决方案。

分析

- 这是一道非常常见的问题。版本控制、配置管理是软件工程中不可或缺的管理机制，因此测试工程师也必须对其十分了解。版本控制在软件开发中的重要作用如下：
 - 作为代码仓库有效的管理软件开发中各个不同版本的源代码和文档，占用空间小并且方便各个版本代码和文档的获取。
 - 对开发小组中对源代码的访问进行有效的协调（不同的版本控制软件采取不同的协调策略）。

常用的版本控制软件有 ClearCase、VS、PVCS 和 Microsoft 公司的 VSS。

例子

例如自动化测试脚本程序、测试用例都需要进行版本控制和配置管理。

- CVS：Concurrent Versions System，协作版本系统或并发版本系统，是一种版本控制系统，主要作用在于方便软件的开发和使用者协同工作。这是个开源工具，最初由 Dick Grune 于 1986 年发布。CVS 基于客户端/服务器结构，构成网络也很方便。
- VSS：VSS 的全称为 Visual Source Safe。作为 Microsoft Visual Studio 中的一名成员，它的主要任务就是负责项目文件的管理。Windows 平台下使用 VSS 开发的典型环境是基于 C/S 架构的，即开发小组中的每个开发者在各自的 Windows 平台下利用开发工具（如 VC）开发项目中的各个模块，而配有专门的服务器集中控制开发过程中的文档和代码。服务器和开发人员的客户机分别装有 VSS 的服务器和客户端程序。

Offer

面试试题 4

Could you please illuminate your understanding of CMMI?

译文

请谈一下你对 CMMI 的理解。

解答

CMMI means Capability Maturity Model Integration as the successor of CMM.

CMMI is developed by Office of the Secretary of Defense (OSD), the Carnegie Mellon Software Engineering Institute (SEI), the National Defense Industrial Association and so on. CMMI is the most advanced quality certification in engineering field.

CMMI is a process improvement approach that provides organizations with the essential elements of effective processes that ultimately improve their performance. CMMI can be used to guide process improvement across a project, a division, or an entire organization.

CMMI aims to integrate different CMM models into one framework in project development. It mainly merges SW-CMM, SECM, IPD-CMM into a uniform frame to provide a process improvement model within the enterprise for an organization.

解答译文

CMMI 是软件能力成熟度集成模型，是 CMM 的改进版，由美国国防部与卡内基-梅隆大学软件工程研究所、美国国防工业协会等多个来自企业、政府的组织研发，是工程领域最高水平的质量认证之一。

CMMI 是一种过程改进的方法，它为组织机构提供了有效过程的基本元素，可以用于指导跨项目、部门或者整个组织的过程改进。

CMMI 的提出主要是为了解决在项目开发中需要用到多个 CMM 模型的问题，它主要结合了 SW-CMM（用于软件的能力成熟度模型）、SECM（系统工程能力模型）、IPD-CMM（集成产品开发能力成熟度模型），将它们融合到一个统一的改进框架内，为组织提供了在企业范围内的进行过程改进的模型。

分析

- 这个问题主要考查对质量管理（Quality Assurance）的理解。CMMI 是对 CMM（软件能力成熟度模型）的改进，它的出现由于工程环境和过程愈加复杂，独立的 CMM 面对更加复杂的要求无法适应。CMMI 有阶段式和连续式两种表示方法，以满足不同用户的需要，尽管表示方法不同，但最终目标是一致的。连续式分 0~5 级，共 24 个过程域。阶段式表示法分 1~5 级，也包含 24 个过程域，如表 8-2 所示。



表 8-2 阶段式 CMMI 能力等级

能力级别	连续式表示能力级别
0	不完整级（Incomplete）
1	已执行级（Performed）
2	已管理级（Managed）
3	已定义级（Defined）
4	已定量管理级（Quantitatively Managed）
5	优化级（Optimizing）

表：连续式 CMMI 能力等级

成熟度级别	阶段式表示成熟度级别
1	初始级（Initial）
2	已管理级（Managed）
3	已定义级（Defined）
4	已定量管理级（Quantitatively Managed）
5	优化级（Optimizing）

- 面试官可能还会问到 CMM 认证、RUP 认证等相关内容。
 - CMM：在美国国防部的支持下，卡内基-梅隆大学软件工程学院（Software Engineering Institute，SEI）于 1991 年发表了 CMM 模型。一般认为，CMM 并不涉及具体的软件流程的细节，仅仅是关注了制定、管理、控制软件流程所必须关注的管理要点。CMM 是指“能力成熟度模型”，是对于软件组织在定义、实施、度量、控制和改善其软件过程的实践中各个发展阶段的描述，其核心就是把软件开发视为一个过程，并根据这一原则对软件开发和维护进行过程监控和研究，以使其更加科学化、标准化，使企业能够更好地实现商业目标。

CMM 是一种用于评价软件承包能力并帮助其改善软件质量的方法，它侧重于软件开发过程的管理及工程能力的提高与评估。CMM 分为 5 个等级：一级为初始级，二级为可重复级，三级为已定义级，四级为已管理级，五级为优化级。其所依据的想法是：只要集中精力持续努力去建立有效的软件工程过程的基础结构，不断进行管理的实践和过程的改进，就可以克服软件生产中的困难。CMM 为软件企业的过程能力提供了一个阶梯式的改进框架，它基于过去所有软件工程过程改进的成果，吸取了以往软件工程的经验教训，提供了一个基于过程改进的框架，同时它又指明了一个软件组织在软件开发方面需要管理哪些主要工作、这些工作之间的关系以及以怎样的先后次序、一步一步地做好这些工作而使软件组织走向成熟。

- RUP：统一软件开发过程（Rational Unified Process），是一个面向对象且基于网络的程序开发方法论。根据 Rational（Rational Rose 和统一建模语言的开发者）的说法，就如同一个在线的指导者，它可以为所有方面和层次的程序开发提供指导方针、模板及事例支持。统一软件开发过程和类似的产品，如面向对象的软件过程（OOSP）及

Offer

OPEN Process 都是理解性的软件工程工具，把开发中面向过程的方面（例如定义的阶段、技术和实践）和其他开发的组件（例如文档、模型、手册和代码等）整合在一个统一的框架内。

RUP 注重迭代式开发、管理需求、基于组件的体系结构、可视化建模、验证软件质量、控制软件变更。RUP 中的软件生命周期在时间上被分解为 4 个顺序的阶段，分别是：初始阶段（Inception）、细化阶段（Elaboration）、构造阶段（Construction）和交付阶段（Transition）。每个阶段结束于一个主要的里程碑（Major Milestones）；每个阶段本质上就是两个里程碑之间的时间跨度。RUP 中有 9 个核心工作流（Core Workflows），分为 6 个核心过程工作流（Core Process Workflows）和 3 个核心支持工作流（Core Supporting Workflows）。这些工作流在整个生命周期中一次又一次地被访问。

RUP 具有很多优点：RUP 提高了团队生产力，在迭代的开发过程、需求管理、基于组件的体系结构、可视化软件建模、验证软件质量及控制软件变更等方面，针对所有关键的开发活动为每个开发成员提供了必要的准则、模板和工具指导，并确保全体成员共享相同的知识基础，另外它建立了简洁和清晰的过程结构，为开发过程提供较大的通用性。但同时它也存在一些不足： RUP 只是一个开发过程，并没有涵盖软件过程的全部内容，例如它缺少关于软件运行和支持等方面的内容。此外，它没有支持多项目的开发结构，这在一定程度上降低了在开发组织内大范围实现重用的可能性。可以说，RUP 是一个非常好的开端，但并不完美，在实际的应用中可以根据需要对其进行改进。

面试试题 5

What is the difference between Get method and Post method supported by HTTP?

译文

HTTP 协议支持的 Get 方法和 Post 方法有何不同？

解答

1．Transmitted data by Post method are not necessarily texts, while for Get method, only texts can be transmitted.

2．Both of Post and Get are available values of Method(a property of Form). The default value is Get.

3．At server Request.QueryString is always used to catch the date by Get, while Request.Form is always used for data by Post.

4．Post is the securer. Input data by users will be transmitted as a part of URL based upon Get method. However, those will be transmitted independently and will not appear in URL in virtue of Post method.



5. Get limited the object size. Post has no limitation and is adapt to large-size requests.

解答译文

1. 用 Post 传输的数据不一定要是文本，但用 Get 方法传输的却一定要是文本。

2. Post 和 Get 都是表单属性 Method 的可选值，Method 的默认值为 Get。

3. 在服务器端一般用 Request.QueryString 来获取 Get 方式提交来的数据，用 Post 方式提交的数据一般用 Request.Form 来获取。

4. Post 方法更安全。Get 方法中输入的数据将作为 URL 的一部分发送。在 Post 方法中，传输数据时不会将数据作为 URL 的一部分，而是作为一个独立的实体来传输。Post 提交后地栏不变。

5. Get 方法对对象大小有约束，而 Post 方法适于大字段请求。

分析

- HTTP 是超文本传输协议（Hypertext Transfer Protocol），位于应用层，是用于从 WWW 服务器传输超文本到本地浏览器的传送协议。HTTP 定义了信息如何被格式化和传输以及在各种命令下服务器和浏览器所采取的响应。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，而且还确定传输文档中的哪一部分以及哪部分内容首先显示（如文本先于图形）等。基于 HTTP 协议的客户机/服务器模式的信息交换过程分 4 个过程：建立连接、发送请求信息、发送响应信息、关闭连接。
- 虽然 Post 和 Get 两种提交方式可以统一用 Request("oStr")来获取提交数据，但是这样对程序效率有影响，很少被使用。
- 有时 Post 方法传送的数据大小有限制。这是因为：尽管 HTTP 协议规范实际上没有对 URL 长度进行限制，但是特定的浏览器及服务器存在对 URL 长度的限制。
- TCP/IP 四层和 OSI 七层网络模型经常会在面试中被考查，需要大家重点关注。

8.4 项目经验

项目经验一向是面试的重中之重。

在本书第四部分“测试实战篇”中，我们对项目经验做了讲解和总结。在本节第 7 章 7.1 节中，我们给出了 5 道典型的面试题。通过讲解，大家对题型和相关知识有了一定的认识。在测试实战篇中，我们 3 位作者还讲述了自己的一些实战经验，就项目经验面试考查的各个方面进行了讨论。

项目经验是决定面试成败的关键。从我们的经验来看，笔试不如面试重要，面试中的理论考查又不如项目经验的考查重要。例如项目架构、技术特点、模块功能、所遇难点及解法、过程改进、风险控制、测试用例设计以及性能分析等，都是我们要精心准备的重点。

Offer!

本节是用英文来提问和回答有关项目经验的问题，等于是为第四部分“测试实战篇”补充5道面试题。

如何用英文来表述清楚比较复杂的问题是对大家的另一个挑战。这需要读者在英文的写作和口语方面进行刻苦的训练。

小技巧

回答要有层次有条理，用 Firstly、Secondly、furthermore、at last 等表示顺序的插入语来展现层次逻辑。

面试试题 1

How to guarantee high coverage rate by test cases?

译文

如何保证测试用例的覆盖率？

解答

Firstly, test cases should be made according to documents including user requests and so on, those should cover all requirements.

Secondly, unclear descriptions such as “very large”, “good enough” and so forth should not appear in documents. Testers need precise digits.

Thirdly, communicate with development and design departments to prevent from different understanding for obscure places in documents.

Fourth, exploration test should be executed by skilled testing engineers to increase coverage rate.

At last, white-box test should be planned.

解答译文

（1）根据用户需求等文档，保证测试用例覆盖到所有需求。
（2）避免用户需求等文档中有模糊描述，变不可测为可测。
（3）多和开发与设计部门沟通，避免对用户需求等文档的理解不一致。
（4）组织有经验的工程师小组进行探索性测试，以补测试用例的不足。
（5）进行白盒测试。

分析

- 测试用例的覆盖率是测试设计时的一个重要考虑方面。首先，理论上已经证明，测试用例是不可能达到 100%覆盖软件的各个方面。也就是测试只能是不完全测试。其次，根据 80-20 原则，我们也没有任何必要将精力时间平分到各个模块和测试种类来测试，在

进行风险分析后，高风险并且对用户影响巨大的模块和测试种类将被给予更多的关注。

- 有的需求和设计文档上有些模糊内容，如“比较好”、“很好”、“非常不好”等，这样会让测试人员无法测试，因此，我们要求设计和开发人员写明数字化的要求，以供测试时使用。
- 很多时候，开发、设计、测试会对文档某处的理解不同，当测试人员觉得可能有歧义时，要和开发和设计人员沟通，修改文档。
- 测试用例只是覆盖了需求，在需求之外，软件还必须符合通用的一些规则。这些规则很多时候无法覆盖到，因此要进行探索性测试（也叫随机性测试，实则并非随机，而是按照一定的测试模型来测试）。在本书第 3 章第 3.17 小节中，我们给出了一些测试模型。通常会安排一些有经验并且有一定测试理论基础的测试工程师在需求等文档的测试用例之外来进行探索性测试。笔者当年在摩托罗拉公司工作时，曾负责一个探索性测试小组，在测试用例之外发现了大量软件缺陷，提高了测试用例的覆盖率。
- 开发人员通常要进行单元测试等白盒测试，测试人员也应当进行白盒测试。有些软件缺陷在黑盒状态下很难被发现，黑盒有时简直就像盲人摸象，搞不清状况。白盒测试通常是对代码的覆盖，是对黑盒的有益补充，大大提高了覆盖率。

说明

除此之外，我们还应设计一些测试用例，如重要功能、功能接口等，观察代码的逻辑处理。

面试试题 2

If an automatic test case is passed, does this mean that no bugs exist during the execution course?

译文

一条自动化测试用例被测试通过，这是否意味着执行过程中没有 Bug?

解答

Of course not. Since errors may be mutative and cannot be forecasted, it is almost impossible to judge those in the test scripts. Machine cannot completely replace man.

解答译文

当然不是。因为出现的错误千变万化，不太可能在测试脚本中对所有错误进行判断。机器不可能完全替代人工。

分析

- 自动化测试用例的通过与否是由机器判断的。但机器只能依据程序进行判断，而程序中不可能对所有的错误都进行判断。

Offer

说明

可能有人会问，为何不能对所有可能出现的错误都分门别类地进行处理呢？

因为在每个时刻都可能会出现错误。例如忽然一道横线出现在屏幕上时，人工可以轻易地判断，可是机器很难判断，而且测试脚本是为了替代人工而写的，如果写一个测试脚本包括所有可能性错误，那么写脚本的时间精力会消耗极多，达不到之前替代人工的目的了。

- 这个问题的提出实际上是指出了自动化测试常见的缺点。通过这个问题可以考查出面试者是否真的写过测试脚本。

面试试题3

Please talk about the understanding of automation function-test, and the methods to do automation function-test.

译文

请谈谈对自动化功能测试的理解，并说出自动化功能测试的方法有哪些？

解答

Automation function-test is the use of software to control the execution of tests, the comparison of actual outcomes with predicted outcomes, the setting-up of test preconditions, and other test control and test reporting functions. Commonly, test automation stands for automating a manual process already in use.

The methods of automation function-test:

1. Code-driven testing: Usually, public interfaces to classes, modules, or libraries are tested with a variety of input arguments to validate whether the results returned are correct.

2. Graphical user interface testing: A testing framework generates user interface events such as keystrokes and mouse clicks, observes the changes that happen in the user interface, to validate whether these observable behaviors of the program are correct.

解答译文

自动化功能测试就是使用软件来控制测试的执行，对实际输出和预期输出进行对比，测试前提条件的构建，以及其他测试控制条件和测试报告功能。通常，测试自动化就是对一个已经使用的手工测试过程的自动化。

自动化功能测试方法主要有以下两种。

（1）代码驱动的测试：主要是对类、模块或库文件的公共接口进行测试，通过输入多种不同类型参数来验证返回结果的正确性。

（2）图形用户界面测试：使用测试框架生成用户界面事件，例如按键和鼠标点击，并观察在用户界面的变化，以验证程序的这些可观察行为的正确性。

Offer

说明

自动化测试工具一般都很昂贵，并且通常都是和手工测试配合使用的。如果自动化功能测试工具可以在很长一个时期内（回归测试至少要做4次）被使用，就可以很好地降低成本。

面试试题4

If a product will be on the market tomorrow, what will you do to test the product?

译文

如果产品明天就要上市了，你将如何来测试这个产品？

解答

We'd like to arrange a user trial test. We'll involve as many customers as possible to do this test, because the product is going to be on the market. For the time issue, we have no opportunity to do any more interval systematical test on it. Since our target is to deliver this product to customers, why don't we let them find out their favorite parts of the products and catch bugs in those parts for us?

解答译文

我们可以组织一次用户体验测试。因为产品即将上市，我们将引入尽可能多的用户来做这个测试。因为时间问题，我们没有机会去做更多内部的系统化测试。既然我们的目标是为了将此产品发布给客户，那我们为什么不让他们找到产品中自己最喜欢的部分并为我们寻找缺陷呢？

分析

根据20-80原则，在通常情况下，只有20%的软件功能会被80%以上的用户经常使用（我们自己肯定也有相似的体会）。那么谁最了解那20%是什么功能呢？当然是用户自己，所以我们可以尽可能地组织大量的目标用户对软件产品进行测试。这样，样本的数量越大，能够正确圈定80%以上的用户经常使用的20%的软件功能的概率也就越大。

说明

当然，作为用户体验测试来说，测试人员还担负着选定用户体验用户的人选、分发产品、建立用户递交软件问题的通道和从用户所提交的软件问题中筛选软件缺陷等任务。

面试试题5

How long do you finish a test plan document?

译文

你完成一个测试计划文档需要多长时间？

解答

Actually, it takes me the time from the document initialization to the end of the test activity.

解答译文

实际上，它耗费我整个从起草该文档一直到测试活动结束的时间。

分析

实际上，软件测试计划文档中的内容是随着软件测试的行进动态变化的。例如关于测试任务量估计章节，就需要测试计划文档起草人员在最初时根据以往的软件测试的项目经验预估整个项目的工作量。但是当项目进行一段时间后，测试和开发人员对产品的各方面特性和技术复杂度有一定的认识之后，势必会对此后的未完成工作量估计做出一定的调整。一旦未完成工作量估计做出调整了，产品的时间规划和任务分配也就会随之调整。类似于这样的调整在整个测试活动中都会遇到。因此修改和维护计划文档中的内容中的内容是贯穿于整个测试活动的，不应该是写完一遍就束之高阁。

8.5 软件测试面试英文、能力、心理测试

在面试笔试题中，除了考查软件测试概念与原理、相关计算机领域（包括质量管理）、项目经验之外，还有对英文水平、性格、心理、智力及能力的考题等。

这里面有的问题是有固定答案的。对于那些没有固定答案的问题，回答时一定要小心，这里面很可能有“陷阱”面试时应聘者应务必注意言多必失这个道理，因此老套的回答至少不冒风险。当然，这是个人风格的问题。有的人敢于冒险，也许让面试官耳目一新，留有深刻印象。

然而，这部分回答得再精彩，也没有决定性的作用。起决定作用的还是前面的软件测试概念与原理、相关计算机领域（包括质量管理）、项目经验。当然，英文能力也相对重要。说实话，若理论实践和经验都十分优秀，则英文略差一些也无妨。

8.5.1 翻译

翻译题是常用的考查英文水平的题型，通常来说就是几段文字的中英文互译，具有大学英语四级的水平足以解答。难点在于文字中涉及一些专业词汇。通常的笔试类一般采用开卷考试，无人监考也可以看资料，因此如果对方允许，还可以使用电子辞典。

面试试题1

Please translate this paragraph:



For this reason, development team shifted the focus of solving defects from type Worksforme to type Later, recognizing that some puzzlement needs to wait for more detailed information from testers. If the project process observed RUP method, we would expect the opposite. Reusable scripts make sense as well in new version as they did in old one. In our company it used to be that 1/3 of team members are experienced testers, now close to 1/2 are.

解答

因此，由于认识到一些疑难问题需要测试人员给出更进一步的详细信息，开发团队将修改缺陷的焦点从 Worksforme 类型的缺陷转向了 Later 类型的缺陷。如果项目过程遵循了 RUP 模式，则本应看到相反的结果。和在旧版本中一样，可复用的脚本在新版本中也很有用。以前在我们公司，1/3 的测试团队成员是有经验的测试人员，现在这个数字已接近 1/2。

分析

- 第一句涉及了开源缺陷跟随系统 Bugzilla。Bugzilla 将缺陷分类，Worksforme 类指的是开发人员看不懂测试人员关于这个缺陷的描述，或者不能重现这个缺陷。这意味着需要测试人员给出进一步的详细信息。Later 类指的是开发人员认同测试人员提交的这个缺陷确实存在，但是不在这个软件版本中解决。本句描述的情况便是在升级版本中。
- For this reason，意为：因此。此原因是在前文中已指明的。
- Recognizing + that 从句，在全句中做状语，主语同主句，是 development team。
- shift the focus from A to B，意为：将焦点由 A 移至 B。

puzzlement，不可数名词，意为：迷惑。

- 第二句意指，因为没看到相反的结果，所以项目过程未遵循 RUP 模式。
- RUP，即 Rational Unified Process，Rational 统一过程。它是美国 Rational 软件公司开发的一套软件过程框架，帮助软件企业的软件开发过程管理控制有章可循，以提高软件质量。
- Would，will 的过去时，意为：本应。从句和主句的时态都是过去时，因为都是过去发生的动作。
- opposite，名词，意为：相反的事物。
- 第三句中的 Reusable，形容词，意为：可复用的，可重用的。
- make sense，意为：起作用，有意义。
- Did，代指 made sense，避免重复。
- One，代指 version，避免重复。
- 第四句中的 It used to be + that 从句，意为：过去常常……
- 后一句省略了两处，复原为：now close to 1/2 of team members are experienced testers.之所以省略，是为了避免与前句重复。

面试试题 2

Please translate these Chinese sentences into English:

Offer!

测试应追溯到用户需求，因为最后将基于合同进行项目验收。不可测试的功能性和非功能性需求会对系统的成功构成巨大威胁。把系统在正常条件下的性能看作是长时间大负载下的性能是不可靠的。一直认为过程管理至少不比极限编程差的人数少于预期。

解答

Test should be traced back to the user requests by the realization that in the end checking and accepting the project is based on the contract. Functional and non-functional requests without mensurability will jeopardize the future of a successful system. It is unlikely that performance of the system in a normal situation is representative of that with heavy load for a long time. Fewer people than expected have long assumed that process management is almost as good as, if not better than, eXtreme Programming.

分析

- 第一句中的 trace back to sth.，意为：追溯到某事。
- that 引导的从句，是 realization 的同位语从句。
- be based on sth.，意为：基于某事物。
- 第二句主干是：Requests will jeopardize the future。
- non-functional，复合形容词，意为：非功能的。non-，前缀，表示“非，无，不”。
- Mensurability，名词，意为：可测量性。
- Jeopardize，动词，意为：危害。
- Future，名词，意为：前景。
- 第三句不是强调句型。全句主干为 It is unlikely + that 从句。It 是形式主语，that 从句是真正的主语。
- Unlikely，虽有 ly，但是为形容词，意为：未必的，不太可能的，靠不住的。
- in a normal situation，意为：在正常条件下。
- Representative，名词，意为：代表，典型。A is representative of B，意为：A 代表 B。
- That with heavy load for a long time 中的 that，指代前面的 performance，避免重复。
- 第四句中的 Fewer people than expected，有省略。补全的话是：Fewer people than the people which are expected。由于 people 和前面重复，故省略 the people，同时还省略了 which are。
- if not better than 做插入语。
- eXtreme Programming，缩写为 XP，极限编程。相比于过程管理思想，XP 理论是采取简单方式进行规划和跟踪，以成对编程和团队整体进行工作，适合于用户需求变化不大的小项目。

8.5.2 性格心理考查

看似随意的聊天，但面试官会从中归纳总结很多对应聘者的看法。



面试试题3

Why do you want to join this company?

译文

你为何希望来我们公司？

解答

With no doubt, money is important but more important for me, here I can better recognize my personal value.

解答译文

我不否认工资是个很重要的方面。但是更重要的，我相信在这家公司能更好地实现我的价值。

分析

- 对此问题最糟糕的回答就是："Because you could provide higher salary（你们给的工资高）"。虽然90%是这个原因，但是在这个世界上很多事情需要包装，太直白会让别人接受不了。因此如果你没有特别的原因，可以像以上这样回答。
- 老套的回答没有新意，但是保险，面试官不会有很多联想。标新立异的回答，有吸引力，但会有风险。其实关键还在于应聘者实力。在没有实力的情况下，面试官可以对应聘者很挑剔；当你的实力很强时，只要面试期间没有出格的、严重的错误行为，回答得标新立异一点又有什么关系呢？

8.5.3 智力考查

很多种工作的面试题中都有智力题。例如笔者曾面试过英文编辑、中文编辑，都碰到了这种题型。实际上，这种题答对与否对面试的影响并不大。因为绝大部分工作都是熟练工种，如软件测试，频繁动脑以至于绞尽脑汁的情况不太可能出现。

面试试题4

There is a cake in a box. Now you need cut the cake into 8 pieces and give to 8 persons, while 1 piece should be left in the box. What will you do?

译文

请把一盒蛋糕切成8份，分给8个人，但蛋糕盒里还必须留有一份。

解答

Divide the cake into 8 pieces. Give 7 pieces to 7 people respectively, give the box and the last

piece(in the box) to the 8th person。

解答译文

把切成的 8 份蛋糕先拿出 7 份分给 7 人，剩下的 1 份连蛋糕盒一起分给第 8 个人。

分析

思维突破的关键点是：盒子也可以给人。这道题不难，其实就是道脑筋急转弯。当然，你也许会遇到需要复杂逻辑分析的智力题。

8.5.4 能力考查

能力考查通常是一些开放性问题，没有固定的答案。从应聘者的回答中，面试官能看出应聘者处理问题的风格。在回答这种问题时，最好要对自己的方式、采取的步骤做出合乎情理的解释。

面试试题 5

If your superior quarrels with your subordinate, how do you deal with this problem?

译文

如果你的上级和下级争吵，你怎么处理？

解答

If this quarrel is about a technical issue, I need analyze the issue in depth and then support the debate to find the correct answer. At the same time I would work to prevent this debate from being more than technique. If the quarrel roots from some personal abuse, I would seek who is wrong in this incident based on justice and truth. After that, Chinese people think highly of "balance", I would company with the offender to say sorry to the other side in a private appointment. If the mistake is quite serious, I would require a public apology. Moreover, I would warn workmates to not disseminate this incident。

解答译文

如果是单纯技术上的争论，我们要仔细思考，支持辩论直到清楚，并且协调保证这种争论不要转向人身攻击；如果是非技术的私人争执，中国人讲究“中庸”，我们要讲道理，看看错在哪方，然后选择私下方式与双方交流，让错误的一方向另一方道歉，并约束同事们不要大肆宣传。

分析

这是一个典型的开放型问题。处理这种难题的方式多种多样。以上解答只是其中的一种。回答的要点是：处理时要虑及长远，不能不讲理，不能以势压人，不能恃强凌弱。

Offer

第9章 软件测试常用文档模板

9.1 测试计划模板

软件生命周期模型已由瀑布模型发展为螺旋模型，因此测试不再是一个独立的片段，而是贯穿于整个软件开发过程之中。

在表 9-1 中，相应阶段可以同步进行相应的测试计划编制，而测试设计也可以结合在开发过程中实现并行，测试的实施即执行测试的活动即可连贯在开发之后。值得注意的是，由于单元测试和集成测试往往由开发人员承担，因此这部分的阶段划分可能会安排在开发计划而不是测试计划中，如表 9-1 所示。

表 9-1　　测试计划按照阶段及类型分布

阶段 / 测试类型	需求	设计	编码	单元	集成	系统	确认
确认测试		设计					执行
系统测试		设计				执行	
集成测试			设计		执行		
单元测试			计划/设计	执行			

测试计划中强调对变更的控制，这里的变更主要指项目计划、需求、测试产品版本、测试资源的变更。测试阶段的风险主要表现在对上述变更所造成的不确定性，有效地应对这些变更就能降低风险。

在测试过程中，通常是由于时间紧张的原因，很可能需要对测试计划进行调整。调整的目的是重新检查不重要的测试部分，调换测试的次序和减少测试规模，对测试类型重新组合择优，力求在限定时间内做最重要部分的测试，这样就可以把忽略部分留给确认测试或现场测试。

好的测试计划是成功的一半，而另一半则是对测试计划的执行，因此另一个极其重要的问题就是对测试计划的管理和监控。两者结合才能保证产品的质量，才能实现测试计划的最终目标。

请参考以下的测试计划模板。

Offer

9.1.1 中文模板

（封面）

产品名称	密　级	
产品版本	共	XX 页

（封 1）

测试计划

拟　制		日　期	
评 审 人		日　期	
批　准		日　期	

修订记录

日　期	修订版本	需求更改/缺陷号	修改章节	修改描述	作　者

目录

XXXXXXXXXXXXX

XXXXXXXXXXXXX

XXXXXXXXXXXXX

表目录

XXXXXXXXXXXXX

XXXXXXXXXXXXX

XXXXXXXXXXXXX

图目录

XXXXXXXXXXXXX

XXXXXXXXXXXXX

XXXXXXXXXXXXX

（具体内容）

（封 1）

测试计划

关键词：

XXXXXXXXXXXXX。

摘　　要：

XXXXXXXXXXXXX。

缩略语清单：

缩 略 语	英文全名	中文解释

1．简介

1.1 目的

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

1.2 范围

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2．测试计划

2.1 资源需求

2.1.1 软件需求

软件需求表

资　　源	描　　述	数　　量

2.1.2 硬件需求

硬件需求表

资　　源	描　　述	数　　量

2.1.3 其他设备

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.1.4 人员需求

Offer!

人员需求表

资　源	技能级别	数　量	到位时间	工作期间

2.2 测试对象

2.2.1 测试对象依赖关系

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.3 过程条件

2.3.1 启动条件

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.3.2 结束条件

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.3.3 挂起条件

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.3.4 恢复条件

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.4 进度计划

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.5 导向/培训计划

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

3．测试方法

3.1 针对测试实体，给出测试的设计思路和方法

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

3.2 测试组网图

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

3.3 测试顺序

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

4．工作交付

工作交付列表

名　称	作　者	应交付日期

5．参考资料清单



XXXXXXXXXXXXX
XXXXXXXXXXXXX
XXXXXXXXXXXXX

9.1.2 表格项解释

（1）缩略语清单：对本文所用缩略语进行说明，要求提供每个缩略语的英文全名和中文解释。

（2）目的：这部分要描述文档的目的。应对读者指明。

（3）范围：定义文档适用和不适用的范围，同时应描述计划所覆盖的模块、子项目及项目。

（4）软件需求：本节描述测试所需要的支撑软件的名称、数量、版本，如操作系统、数据库、编译器、预处理器和测试工具等，同时还应描述每一项的用途及应用范围。

（5）硬件需求：本节描述测试所需要的硬件环境的名称、数量、版本，如硬件设备、计算机、打印机、接口设备、测试仪器以及各种软件运行所需的硬件设备等，同时还应描述每一项的用途及应用范围。

（6）其他设备：本节描述测试中所需用到的其他材料，如用户手册、操作指南和相关工业标准等，同时还应描述每一项的用途及应用范围。如果没有可忽略此节。

（7）人员需求：本节表述人员相关的其他信息，包括：特殊要求、关键技能、领域经验和外部支持等。如果需要，还应描述人员相关的其他信息，如轮班制度等。

（8）测试对象：标识出所要测试的对象和范围。

（9）测试对象依赖关系：本节描述被测对象间的关系、被测对象与软件其他部分间的关系，并且注明各个元素之间的依赖关系，以确定其测试顺序。

（10）启动条件：本节测试执行活动开始所应具备的条件，包括：测试用例与测试对象的基线化，所需的测试环境（包括硬件环境、软件环境、相关依赖模块、组网等）和测试代码（测试桩、测试驱动等）的准备就绪，测试脚本，测试数据已经生成。

（11）结束条件：本节测试结束的条件或标准。例如满足测试质量目标比如测试覆盖率已经达到，遗留问题的个数为零，缺陷分析中没有发现重大问题，以及所有测试用例已经通过等。

（12）挂起条件：测试过程无法继续时应挂起测试，本节描述会导致测试挂起的各种因素。这些因素可能包括：测试依赖因素不具备、进入某项风险临界区及管理决策等。

（13）恢复条件：本节描述测试挂起后恢复测试的条件，与挂起条件相对应。

（14）进度计划：给出测试持续时间和到进度文件的链接。

（15）导向/培训计划：本段描述测试前或测试中需要进行的指导、培训工作。这些培训可以包括用户指引、操作指引、维护控制组指引，对成员的指导简报。如果需要提前进行大规模培训，则应将培训计划作为项目计划一个独立的部分，并在此处引用。

（16）测试组网图：为特定的测试活动准备相应的组网图，并注明组网应用范围。

（17）测试顺序：给出所测试的函数、模块或者子系统的测试顺序。

Offer!

9.1.3 英文模板

（Cover）

Product name	Confidentiality level	
Product version	Total	xx pages

(Inside front cover)

Test Plan

Prepared by		Date	
Reviewed by		Date	
Approved by		Date	

Revision Record

Date	Revision Version	CR ID / Defect ID	Sec No.	Change Description	Author

Catalog

XXXXXXXXXXXXX

XXXXXXXXXXXXX

XXXXXXXXXXXXX

Table List

XXXXXXXXXXXXX

XXXXXXXXXXXXX

XXXXXXXXXXXXX

Figure List

XXXXXXXXXXXXX

XXXXXXXXXXXXX

XXXXXXXXXXXXX

(Content)

Test Plan

Keywords:

XXXXXXXXXXXXX。

Abstract:

Offer

XXXXXXXXXXXXX。

List of abbreviations:

Abbreviations	Full spelling	Explanation

1. Introduction

1.1 Purpose

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

1.2 Scope

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2. Test Plan

2.1 Resource Requirements

2.1.1 Software Requirements

Software Requirements Table:

Resource	Description	Qty

2.1.2 Hardware Requirements

Hardware Requirements Table2

Resource	Description	Qty

2.1.3 Other Materials

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.1.4 Personnel Requirements

Personnel Requirements Table

Resource	Skill Level	Qty	Date	Duration

Offer

2.2 Test Object

2.2.1 Object Dependency

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.3 Process Criteria

2.3.1 Entry Criteria

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.3.2 Exit Criteria

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.3.3 Suspend Criteria

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.3.4 Resume Criteria

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.4 Schedule

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

2.5 Orientation/Training Plan

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

3.Test Method

3.1 Describe the design idea of test for test entities

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

3.2 Test Topologies

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

3.3 Test Flow

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX。

4.Deliverables

Deliverables Table

Name	Author	Delivery Date

5. List of reference

XXXXXXXXXXXXX

XXXXXXXXXXXXX

XXXXXXXXXXXXX



9.2 性能测试计划模板

性能是衡量软件质量的主要标准之一，尤其体现在与 Web 相关的系统上，例如目前比较主流的 B/S 架构的软件系统。性能测试对保证软件质量有着重要作用，主要体现在以下几个方面。

- 评估系统的能力，测试中得到的负荷和响应时间数据可以被用于验证所计划的模型的能力，并帮助做出决策。
- 识别体系中的弱点：受控的负荷可以被增加到一个极端的水平，并突破它，从而修复体系的瓶颈或薄弱的地方。
- 系统调优：重复运行测试，验证调整系统的活动得到了预期的结果，从而改进性能。检测软件中的问题：长时间的测试执行可导致程序发生由于内存泄露引起的失败，揭示程序中的隐含的问题或冲突。
- 验证稳定性（resilience）可靠性（reliability）：在一个生产负荷下执行测试一定的时间是评估系统稳定性和可靠性是否满足要求的唯一方法。

性能测试通过自动化的测试工具模拟多种正常、峰值及异常负载条件来对系统的各项性能指标进行测试，例如针对.NET 和 J2EE 架构的 Web 系统的性能测试。

性能测试要求对系统性能进行全面的分析和探测系统瓶颈，从而帮助开发人员改进算法、架构和功能编码，使系统更加完善。

性能测试人员需要对数据库进行初步性能调优，对网络服务器进行参数配置调优，另外还要和数据库管理员、网络管理员及开发人员紧密配合，这要求性能测试人员具备很强的综合分析问题的能力。

性能测试主要包括 5 个方面。

- 预期目标用户测试。
- 用户并发测试：包括核心模块并发测试、组合业务并发测试。
- 疲劳强度与大数据量测试：包括疲劳强度测试、大数据量测试。
- 网络性能测试。
- 服务器性能初级测试。

对以上 5 个方面的具体解释参见本书第 9.6 节的“性能用例模板”一章中的概述部分。

B/S 或 C/S 等具体的软件架构在实际测试中制约着测试策略和工具的选择。

性能测试项目步骤如下。

（1）制定目标和分析系统。

（2）选择测试度量的方法。

（3）学习的相关技术和工具。

（4）制定评估标准。

Offer!

（5）设计测试用例。

（6）运行测试用例。

（7）分析测试结果。

性能测试团队的角色名称与主要职责如表 9-2 所示。

表 9-2 性能测试团队角色与职责

角色名称	主要职责
项目经理	确定性能测试策略与目标，制定性能测试计划。负责测试工作的整体协调，与开发、支持等其他团队成员进行协作和沟通，保证性能测试工作的顺利进行
系统架构设计师	协助测试设计人员进行系统功能分析，找出系统的压力点，尤其是分析压力下核心业务模块的运行情况
测试设计人员	分析性能测试任务，根据性能测试需求设计相关的测试用例
测试工具专家	使用测试工具开发测试脚本、创建测试场景、运行测试、实时监控数据等，并提供测试结果数据，协助其他人员进行性能分析
数据库系统分析员	根据测试结果数据分析数据库瓶颈，并对数据库进行系统性能调优
Web Server 系统分析员	根据测试结果数据分析 Web Server 瓶颈，并对 Web Server 进行系统性能调优
操作系统专家	为测试提供支持。主要是分析操作系统的运行情况，发现操作系统瓶颈并进行优化

测试计划树立了一个明确的测试目标，明确了需求，组合了各种有针对性的测试方法，明确了里程碑及结束标志，考虑了风险，这样就使得我们对整个项目有了一个清晰的把握。测试计划对于性能测试极为重要。下面是性能测试计划的中英文模板。

9.2.1 中文模板及表格项解释

性能测试计划

1．文档信息

项目名称			
项目经理		文档版本编号	
Focus PM 阶段		文档版本日期	
质量复审方法			
起草人		起草日期	
复审人		复审日期	

2．版本记录

版本编号	版本日期	创建/修改者	说　明

3．引言

XXXXXXXXXXXXX。

4．编制目的

XXXXXXXXXXXXX。

5．性能目标

XXXXXXXXXXXXX。

6．参与者

XXXXXXXXXXXXX。

7．预期读者

XXXXXXXXXXXXX。

8．工具使用

XXXXXXXXXXXXX。

9．培训需要

XXXXXXXXXXXXX。

10．资金预算

对设备及人员需要支付资金的预算。

11．术语定义

XXXXXXXXXXXXX。

12．参考资料

名　称	编 制 者	编制时间

13．人员及职责

13.1 人员组成

测试人员	
开发人员	

13.2 职责

测试人员：开发测试脚本、执行测试，分析测试结果，主要如下：

XXXXXXXXXXXXX。

开发人员：配合进行处理一些在影响测试的代码，主要如下：

XXXXXXXXXXXXX。

14．测试目标、内容、方法、环境

14.1 测试目标

XXXXXXXXXXXXX。

14.2 测试内容

XXXXXXXXXXXXX。

14.3 测试方法

Offer!

XXXXXXXXXXXXX。

14.4 测试环境

14.4.1 硬件环境

- 服务器端

A）数据库服务器

CPU	
MEM	
Disk	
IP(外)	
IP(内)	

B）应用服务器

CPU	
MEM	
Disk	
IP(外)	
IP(内)	

- 客户端

A）客户端 1

CPU	
MEM	
Disk	
IP	

B）客户端 2

CPU	
MEM	
Disk	
IP	

14.4.2 软件环境

- 服务器端

A）数据库服务器

操作系统

数据库

B）应用服务器

操作系统

应用软件

- 客户端

操作系统

Offer!

软件

15. 测试中需提交的文档

XXXXXXXXXXXXX。

16. 测试中需监控内容

- 每次测试中的并发用户数。
- 应用主机性能：CPU 利用率、MEM 利用率、磁盘 I/O 情况。
- Web 系统监测参数。
- XXXXXXXXXXXXX。
- XXXXXXXXXXXXX。

17. 进度安排

日　期	责 任 人	任　务

18. 测试场景 & 用例设计

18.1 预期目标用户测试

18.1.1 测试目的

XXXXXXXXXXXXX。

18.1.2 模式用户行为

- 步骤 1：XXXXXXXXXXXXX。
- 步骤 2：XXXXXXXXXXXXX。
- 步骤 3：XXXXXXXXXXXXX。
- 步骤 4：XXXXXXXXXXXXX。

18.1.3 测试方法&场景设计

XXXXXXXXXXXXX。

18.1.4 测试采集数据

请参考性能测试用例模板。

18.2 核心模块并发测试

18.2.1 测试目的

XXXXXXXXXXXXX。

18.2.2 模式用户行为

- 步骤 1：XXXXXXXXXXXXX。
- 步骤 2：XXXXXXXXXXXXX。
- 步骤 3：XXXXXXXXXXXXX。
- 步骤 4：XXXXXXXXXXXXX。

Offer

18.2.3 测试方法&场景设计

XXXXXXXXXXXXX。

18.2.4 测试采集数据

- 应用系统数据

并发用户数	步骤 1			步骤 2			步骤 3			步骤 4		
	最大	最小	平均	最大	最小	平均	最大	最小	平均	最大	最小	平均

- 硬件数据

并发用户数与应用主机的关系表

并发用户数	CPU 利用率						内存利用率									磁盘利用率		
	%Processor Time			%User Time			可用量(Mbytes)			页数/s			缺页中断数/s			最大	最小	平均
	最大	最小	平均	最大	最小	平均	最大	最小	平均	最大	最小	平均	最大	最小	平均			

- 网络服务器数据

并发用户数	网络服务器								
	CPU 使用率			HTTP 请求数/s			发送量(Kbytes/s)		
	最大	最小	平均	最大	最小	平均	最大	最小	平均

18.3 组合业务并发测试

18.3.1 测试目的

XXXXXXXXXXXXX。

18.3.2 模式用户行为

- 步骤 1：XXXXXXXXXXXXX。
- 步骤 2：XXXXXXXXXXXXX。
- 步骤 3：XXXXXXXXXXXXX。
- 步骤 4：XXXXXXXXXXXXX。

18.3.3 测试方法&场景设计

XXXXXXXXXXXXX。

Offer

18.3.4 测试采集数据

请参考性能测试用例模板。

18.4 疲劳强度测试

18.4.1 测试目的

XXXXXXXXXXXXX。

18.4.2 模式用户行为

- 步骤 1：XXXXXXXXXXXXX。
- 步骤 2：XXXXXXXXXXXXX。
- 步骤 3：XXXXXXXXXXXXX。
- 步骤 4：XXXXXXXXXXXXX。

18.4.3 测试方法&场景设计

XXXXXXXXXXXXX。

18.4.4 测试采集数据

请参考性能测试用例模板。

18.5 大数据量强度测试

18.5.1 测试目的

XXXXXXXXXXXXX。

18.5.2 模式用户行为

- 步骤 1：XXXXXXXXXXXXX。
- 步骤 2：XXXXXXXXXXXXX。
- 步骤 3：XXXXXXXXXXXXX。
- 步骤 4：XXXXXXXXXXXXX。

18.5.3 测试方法&场景设计

XXXXXXXXXXXXX。

18.5.4 测试采集数据

请参考性能测试用例模板。

18.6 网络性能测试

18.6.1 测试目的

XXXXXXXXXXXXX。

18.6.2 模式用户行为

- 步骤 1：XXXXXXXXXXXXX。
- 步骤 2：XXXXXXXXXXXXX。
- 步骤 3：XXXXXXXXXXXXX。
- 步骤 4：XXXXXXXXXXXXX。

18.6.3 测试方法&场景设计

Offer

XXXXXXXXXXXXX。

18.6.4 测试采集数据

请参考性能测试用例模板。

18.7 服务器性能初级测试

18.7.1 测试目的

XXXXXXXXXXXXX。

18.7.2 模式用户行为

- 步骤 1：XXXXXXXXXXXXX。
- 步骤 2：XXXXXXXXXXXXX。
- 步骤 3：XXXXXXXXXXXXX。
- 步骤 4：XXXXXXXXXXXXX。

18.7.3 测试方法&场景设计

XXXXXXXXXXXXX。

18.7.4 测试采集数据

请参考性能测试用例模板。

19.软件测试完成标准

XXXXXXXXXXXXX。

20.局限性

XXXXXXXXXXXXX。

21.测试风险

XXXXXXXXXXXXX。

9.2.2 英文模板

Performance Test Plan

1. Document Information

Project Name			
Project Manager		Document Version Number	
Focus PM Phase		Document Version Date	
Quality Review Method			
Draftsman		Draft Date	
Reviewer		Review Date	

2. Version Record

Version Number	Version Date	Creator/ Mender	Comments

Offer

3. Introduction

XXXXXXXXXXXXX。

4. Objective

XXXXXXXXXXXXX。

5. Performance Goals

XXXXXXXXXXXXX。

6. Participants

XXXXXXXXXXXXX。

7. Main Audience

XXXXXXXXXXXXX。

8. Tools Used

XXXXXXXXXXXXX。

9. Training Needs

XXXXXXXXXXXXX。

10. Budget

XXXXXXXXXXXXX。

11. Term Definition

XXXXXXXXXXXXX。

12. Reference

Material Name	Compiler	Compilation Date

13. Team Members and Responsibilities

13.1Team Members

Testers	
Developers	

13.2Responsibilities

Testers: Develop test scripts, perform tests, analyze test results. Their responsibilities are listed as below:

XXXXXXXXXXXXX.

Developers: Assist to deal with codes influenced tests. Their responsibilities are listed as below:

XXXXXXXXXXXXX.

14. Test Objective, Content, Method and Environment

14.1Test Objective

XXXXXXXXXXXXX.

Offer!

14.2Test Content

XXXXXXXXXXXXX。

14.3Test Method

XXXXXXXXXXXXX。

14.4Test Environment

14.4.1Hardware Environment

- Server

A）Database Server

CPU	
MEM	
Disk	
IP(External)	
IP(Internal)	

B）Application Server

CPU	
MEM	
Disk	
IP(External)	
IP(Internal)	

- Client

A）Client1

CPU	
MEM	
Disk	
IP	

B）Client2

CPU	
MEM	
Disk	
IP	

14.4.2Software Environment

- Server

A）Database Server

Operation System:

Database:

Offer

B）Application Server

Operation System:

Application Software:

- Client

Operation System:

Software:

15. Test Deliverables

XXXXXXXXXXXXX。

16. Monitored Contents during Testing

XXXXXXXXXXXXX。

17. Schedule

Date	Executant	Assignment

18. Test Scenarios & Case Design

18.1Anticipant Objective Test

18.1.1Test Objective

XXXXXXXXXXXXX.

18.1.2Simulation User Actions

- Step 1：XXXXXXXXXXXXX.
- Step 2：XXXXXXXXXXXXX.
- Step 3：XXXXXXXXXXXXX.
- Step 4：XXXXXXXXXXXXX.
- 18.1.3Test Methods & Scenario Design

XXXXXXXXXXXXX.

18.1.4Test Sample Data

Please refer to 18.2.4 and the chapter-*performance test case template* of this book.

18.2Concurrent Core Module Test

18.2.1Test Objective

XXXXXXXXXXXXX.

18.2.2Simulation User Actions

- Step 1：XXXXXXXXXXXXX.
- Step 2：XXXXXXXXXXXXX.
- Step 3：XXXXXXXXXXXXX.

Offer

- Step 4：XXXXXXXXXXXXX.

18.2.3Testing Methods & Scenario Design

XXXXXXXXXXXXX.

18.2.4Test Sample Data

- Application System Data

Current Users	Step 1			Step 2			Step 3			Step 4		
	Max	Min	Ave	Max	Min	Ave	Max	Min	Ave	Max	Min	Ave

- Hardware Data

Concurrent Users and Application Hosts

Current Users	CPU Utilization Percentage						MEM Utilization Percentage									%Disk Time		
	%Processor Time			%User Time			Available Mbytes			Page/sec			Page Faults/sec			Max	Min	Ave
	Max	Min	Ave	Max	Min	Ave	Max	Min	Ave	Max	Min	Ave	Max	Min	Ave			

- Web Server Data

Current Users	Web server								
	CPU Utilization Percentage			Hits/sec			Kbytes Sent/sec		
	Max	Min	Ave	Max	Min	Ave	Max	Min	Ave

18.3Concurrent Combined Businesses Test

18.3.1Test Objective

XXXXXXXXXXXXX.

18.3.2Simulation User Actions

- Step 1：XXXXXXXXXXXXX.
- Step 2：XXXXXXXXXXXXX.
- Step 3：XXXXXXXXXXXXX.
- Step 4：XXXXXXXXXXXXX.

18.3.3Testing Methods and Scenario Design

XXXXXXXXXXXXX.



18.3.4Test Sample Data

Please refer to 18.2.4 and the chapter-*performance test case template* of this book.

18.4Fatigue Strength Test

18.4.1Test Objective

XXXXXXXXXXXXX.

18.4.2Simulation User Actions

- Step 1：XXXXXXXXXXXXX.
- Step 2：XXXXXXXXXXXXX.
- Step 3：XXXXXXXXXXXXX.
- Step 4：XXXXXXXXXXXXX.

18.4.3Testing Methods and Scenario Design

XXXXXXXXXXXXX.

18.4.4Test Sample Data

Please refer to 18.2.4 and the chapter-*performance test case template* of this book.

18.5Large Data Volume Test

18.5.1Test Objective

XXXXXXXXXXXXX.

18.5.2Simulation User Actions

- Step 1：XXXXXXXXXXXXX.
- Step 2：XXXXXXXXXXXXX.
- Step 3：XXXXXXXXXXXXX.
- Step 4：XXXXXXXXXXXXX.

18.5.3Testing Methods and Scenario Design

XXXXXXXXXXXXX.

18.5.4Test Sample Data

Please refer to 18.2.4 and the chapter-*performance test case template* of this book.

18.6Network performance test

18.6.1Test Objective:

XXXXXXXXXXXXX.

18.6.2Simulation User Actions

- Step 1：XXXXXXXXXXXXX.
- Step 2：XXXXXXXXXXXXX.
- Step 3：XXXXXXXXXXXXX.
- Step 4：XXXXXXXXXXXXX.。

18.6.3Testing Methods and Scenario Design

Offer

XXXXXXXXXXXXX.

18.6.4Test Sample Data

Please refer to 18.2.4 and the chapter-*performance test case template* of this book.

18.7Elementary performance test of servers

18.7.1Test Objective

XXXXXXXXXXXXX.

18.7.2Simulation User Actions

- Step 1：XXXXXXXXXXXXX.
- Step 2：XXXXXXXXXXXXX.
- Step 3：XXXXXXXXXXXXX.
- Step 4：XXXXXXXXXXXXX.

18.7.3Testing Methods and Scenario Design

XXXXXXXXXXXXX.

18.7.4Test Sample Data

Please refer to 18.2.4 and the chapter-*performance test case template* of this book.

19．Implemented Criteria of Entire Performance Test

XXXXXXXXXXXXX.

20．Limitation

XXXXXXXXXXXXX.

21．Risk

XXXXXXXXXXXXX.

9.3 黑盒测试用例中、英文模板

黑盒测试的含义为：已知产品的功能设计规格，进行测试证明每个实现了的功能是否符合要求。

软件的黑盒测试意味着测试要在软件的接口处进行。这种方法是把测试对象看作一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明，因此黑盒测试又称为功能测试或数据驱动测试。黑盒测试主要是为了发现以下几类错误：

- 是否有不正确或遗漏的功能？
- 在接口上输入是否能正确地接受？能否输出正确的结果？
- 是否有数据结构错误或外部信息（例如数据文件）访问错误？
- 性能上是否能够满足要求？



- 是否有初始化或终止性错误？

采用软件测试常用的基本方法包括：等价类划分法、边界值分析法、正交排列法、错误推测法、因果图法、状态图法、测试大纲法、自动测试方法、随机测试方法、敏捷测试及故障模型法。

黑盒测试用例设计方法的选择策略如下。

（1）首先进行等价类划分，包括输入条件和输出条件的等价类划分，将无限测试变成有限测试，这是减少测试量和提高测试效率的最有效办法。

（2）在任何情况下都必须使用边界值分析方法。此方法设计的测试用例发现程序错误的能力最强。

（3）可以用错误推测法追加一些测试用例。

（4）对照程序的逻辑，检查已设计的测试用例的逻辑覆盖度，如果没有达到要求，则应再补充。

（5）如果程序的功能说明中含有输入条件的组合情况，则一开始就可以使用因果图法和状态图法。

（6）对于参数配置类的软件，要用正交排列法选择较少的组合方式达到最佳效果。

（7）对于业务流清晰的系统，可以利用测试大纲法贯穿整个测试案例过程，在案例中综合使用各种方法。

9.3.1 中文模板

项目名称		程序版本	
功能模块名			
编制人		编制时间	
功能特性			
测试目的			
参考信息（参考的文档及章节号或功能项）			
环境要求（测试要求的软、硬件、网络要求）			
特殊规程要求			

用例编号	相关用例	用例说明	输入数据	预期结果	实际结果	测试结果（通过/不通过）	缺陷编号	备注

9.3.2 表格项解释

特殊规程要求：特殊的环境和规程要求。

用例编号：测试用例的序号，通常是模块名+数字。

Offer

相关用例：本用例可能大部分步骤与其他某个用例相同，在此指出这个用例的编号，则在“用例描述”中就可以省略相同的步骤。

用例说明：进行测试的步骤描述。

输入数据：测试用例中使用的数据。

预期结果：逐条与输入项对应，列出预期输出。

实际结果：实际上输出的结果。

测试结果：填写“通过”或“失败”。

缺陷编号：如果用例“失败”，则说明是个可能的缺陷，按照次序给出一个编号。

备注：必要的 Z 附加说明。还可能有测试工程师对失败的用例进行分析，定位原因。另外可能还有附图。

9.3.3 英文模板

Project Name				Program Edition Number				
Function Module Name								
Compiler				Compiling Date				
Function Properties								
Testing Objective								
Reference Info (Documents and Chapters Numbers, Function Items)								
System Requirements (Software, Hardware and Network Qualifications)								
Special Regulations and Requirements								
Testing Case Number	Correlative Cases	Description	Inputted Data	Anticipative Result	Actual Result	Pass/Fail	Defection Number	Annotation

9.4 面向过程的白盒测试用例中、英文模板

白盒测试的含义为：已知产品的内部工作过程，通过测试证明每种内部操作是否符合设计规格要求以及所有内部成分是否经过检查。

软件的白盒测试是对软件的过程性细节做细致的检查。这种方法是把测试对象看作一个打开的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息设计或选择测试用例，对程序

所有逻辑路径进行测试。通过在不同点检查程序状态，确定实际状态是否与预期的状态一致，因此白盒测试又称为结构测试或逻辑驱动测试。白盒测试的目标主要是对程序模块进行如下检查：

- 对程序模块的所有独立的执行路径至少测试一遍。
- 对所有的逻辑判定，取“真”与取“假”的两种情况都能至少测一遍。
- 在循环的边界和运行的界限内执行循环体。
- 测试内部数据结构的有效性。

白盒测试工作量巨大，可视化的工具对于测试效率和质量的提高来说十分重要，例如使用不同的颜色区分已执行和未执行的代码段显示分配内存情况实时图表。

白盒测试是对源代码进行的测试，测试的主要内容包括词法分析与语法分析、静态错误分析、动态检测等。但是对于面向过程的和面向对象的开发语言，测试实现的方式和内容还是存在较大差别的。

对于面向过程的结构化开发方式，主要从覆盖源程序语句的详尽程度分析，逻辑覆盖标准包括以下不同的覆盖标准：语句覆盖、判定覆盖、条件覆盖、条件判定组合覆盖、多条件覆盖和修正判定条件覆盖。这是利用程序的内部逻辑结构设计测试用例，对程序的全部路径或可执行语句达到一定的覆盖率。

一个典型的测试用例应该包括下列详细信息。

a.测试目标。

b.待测试的功能。

c.测试环境及条件。

d.测试日期。

e.测试输入。

f.测试步骤。

g.预期的输出。

h.评价输出结果的准则。

9.4.1 中文模板

编制人		审定人		时间			
项目名称				编号/版本			
测试功能							
参考信息：							
用例编号	用例描述	测试数据	预期结果	实际结果	通过/失败	原因	缺陷编号
环境要求：							

Offer

续表

特殊规程要求：	
用例间的依赖关系：	
覆盖率	
未全覆盖原因	

9.4.2 表格项解释

参考信息：参考的文档及章节号或功能项。

用例描述：进行测试的步骤描述。

测试数据：测试用例中使用的数据。

预期结果：逐条与输入项对应，列出预期输出。

实际结果：实际上输出的结果。

通过/失败：填写“通过”或“失败”。

原因：测试工程师对失败的用例进行分析，并定位原因。

缺陷编号：如果用例“失败”，则说明可能是一个缺陷，按照次序给出一个编号。

环境要求：测试要求的软、硬件及网络要求。

特殊规程要求：特殊的环境和规程要求。

用例间的依赖关系：本用例可能大部分步骤与其他某个用例相同，在此指出这个用例的编号，则在“用例描述”中就可以省略相同的步骤。

覆盖率：依据不同的逻辑覆盖标准，如条件覆盖，写出这些测试用例。这里填写这些测试用例合起来达到这种逻辑覆盖标准的覆盖率。

未全覆盖原因：为何不能达到这种逻辑覆盖标准的100%覆盖？写出原因。

9.4.3 英文模板

Compiler		Approver		Time			
Project Name				Sequence/Version			
Tested Function							
Reference Info:							
Case Sequence	Case Description	Test Data	Anticipated Result	Actual Result	Pass/Fail	Reason	Defect No.
Environment Request:							
Special Regulation Request:							

续表

Dependence among Testing Cases:	
Coverage Percentage	
Why < 100%?	

9.5 面向对象的白盒类测试计划与用例中、英文模板

类是所有面向对象程序的构造基础，对它的测试和传统的面向过程程序的单元测试是不同的，类测试更为复杂。

传统单元测试注重测试单元间接口，而类测试除了测试类中包含的类方法之外，还要测试类的状态，对于此点，传统单元测试中是没有的。

并不是每个类都需要测试和评估。通常有多级子类的父类其测试价值更高。评估时主要依据以下 3 个要素。

- 类在系统中的层次。
- 类本身复杂程度和与外体间的交互复杂程度。
- 开发该类的测试程序所需的成本（时间、人力、其他资源）

测试用例的确定有以下 3 种方式。

- 根据前置条件和后置状态。
- 根据状态转换。
- 根据代码。

其中，根据代码确定测试用例时有以下 3 条规则。

- 所有 Public 方法都需要测试。
- 所有 Private 方法都不需要测试。
- 对于友元方法，需选择测试。

由于状态转换仍旧需要测试者设定特殊值来验证类关联关系，并且很难确定所有的测试用例，所以我们推荐根据前置条件和后置状态来确定测试用例的方式。所谓前置条件是指在执行操作时对系统状态所作的假设，而后置状态是指操作执行完成后系统的状态。

穷举是一种无效率的测试方法，我们选用重要的、覆盖面大的测试用例进行测试。

类测试驱动器是运行一个或多个测试用例的可执行程序，它的编写有以下 3 种方法。

- 调用 main 方法。
- 嵌入静态测试方法。
- 独立测试类方法。

前两者不利于测试代码的复用，交付程序时还需要逐个类地删除测试编码，因此我们推荐使用第 3 种方式。

Offer

测试用例命名公式：测试用例名称 = test + 用例方法名称，例如：对于方法 public void getFrequency()，对应的测试方法为 public void testGetFrequency。

类的测试开始于类的设计阶段，类的说明（在对应 CRC 卡上）也是类测试的重要基准。在迭代开发时，每当发生变化，即应进行回归测试，同时需要重新修改测试用例来适应这些变化。

当根据前置条件和后置状态来确定测试用例时，下面介绍类测试计划（包括用例）模板。

9.5.1 中文模板

类名称		实现者	
跟踪号		测试者	
类描述			
类层次			
风险分析			
确定测试用例策略	根据前置条件和后置状态、测试用例命名公式		
测试用例说明			
交互测试			
构造测试			
功能测试			

测试用例				
用例名	输入		输出	
	前置条件	事件	实际结果	异常

9.5.2 表格项解释

跟踪号：类测试的公司内部编号。

实现者：程序开发人员。

类描述：对类的简单描述。

类层次：对类继承父类、实现接口的情况叙述。

风险分析：评估风险。

测试用例说明：说明要测试的方法的范围等。

交互测试：指明哪个与其他类有交互的方法需要测试。

构造测试：指明哪个构造方法需要测试。

功能测试：指明哪个涉及类的功能的方法需要测试。

描述：本用例测试哪个方法或状态。

事件：在用例对应的原方法中传入各种不同参数。

Offer

异常：异常描述。

9.5.3 英文模板

Class Name		Developer	
Tracing Number		Tester	
Class Description			
Class Levels			
Risk			
Strategy of forming cases			
Test Case Description			
Interaction Test			
Structure Test			
Function Test			

Test Cases				
Case Name	Input		Output	
	Prepositive Condition	Event	Actual Result	Exception

9.6 性能用例模板

性能测试主要包括以下 5 个方面。

- 预期目标用户测试。
- 用户并发测试，包括核心模块并发测试、组合业务并发测试。
- 疲劳强度与大数据量测试，包括疲劳强度测试、大数据量测试。
- 网络性能测试。
- 服务器性能初级测试。

分别解释如下。

1. 预期目标用户测试

系统在需求分析和设计阶段都会提出一些性能指标，这些性能指标将在产品说明书中明确指出，因此完成和这些指标相关的测试是性能测试的首要工作。

2. 用户并发测试

并发性能测试的过程是一个负载测试和压力测试的过程，即逐渐增加负载，直到系统的瓶

颈或不能接收的性能点通过综合分析交易执行指标和资源监控指标来确定系统并发性能的过程。

负载测试是已知性能点的测试，其确定在各种工作负载下系统的性能。目标是测试当负载逐渐增加时，系统组成部分的相应输出项，例如通过量、响应时间、CPU 负载和内存使用等来决定系统的性能。负载测试是一个分析软件应用程序和支撑架构、模拟真实环境的使用，从而确定能够接受的性能的过程。

压力测试是未知性能点的测试，通过确定一个系统的瓶颈或不能接收的性能点来获得系统能提供的最大服务级别的测试。

3．疲劳强度与大数据量测试

疲劳测试是采用系统稳定运行情况下能够支持的最大并发用户数，持续执行一段时间业务，通过综合分析交易执行指标和资源监控指标来确定系统处理最大工作量强度性能的过程。

大数据量测试可以分为两种类型：一是针对某些系统存储、传输、统计、查询等业务进行大数据量的独立数据量测试；二是与压力性能测试、负载性能测试、疲劳性能测试相结合的综合数据量测试方案。

4．网络性能测试

网络应用性能分析的目的是准确展示网络带宽、延迟、负载和 TCP 端口的变化是如何影响用户的响应时间的。

5．服务器性能初级测试

实施测试的目的是实现服务器设备、服务器操作系统、数据库系统、应用在服务器上性能的全面监控，找出系统瓶颈，为调优或提高性能提供依据。

设计测试用例是在了解软件业务流程的基础上进行的，其原则是受最小的影响但提供最多的测试信息，其目标是一次尽可能地包含多个测试要素。这些测试用例必须是测试工具可以实现的，不同的测试场景将测试不同的功能。因为性能测试不同于平时的测试用例，应尽可能把性能测试用例设计得复杂一些，只有这样才有可能发现软件的性能瓶颈。

9.6.1 中文模板及表格项解释

1．预期目标用户测试

用例编号				
性能描述				
用例目的				
前提条件				
特殊的规程说明				
用例间的依赖关系				
步骤	输入/动作	期望的性能（平均值）	实际性能（平均值）	回归测试
（1）				
（2）				
（3）				

2. 用户并发测试

（A）核心模块 A

功能	
目的	
方法	

并发用户数与事务执行情况						
并发用户数	事务平均响应时间（/s）	事务最大响应时间	平均每秒处理事务数	事务成功率（/s）	每秒点击率	平均流量（B/s）

并发用户数与数据库服务器					
并发用户数	CPU 利用率	内存利用率	磁盘 I/O 参数	数据库参数	其他参数

并发用户数与应用服务器			
并发用户数	CPU 利用率	内存利用率	磁盘 I/O 系数

（B）组合业务

功能	
目的	
方法	

并发用户数与事务执行情况														
并发用户数	事务平均响应时间（秒）			事务最大响应时间（秒）			平均每秒处理事务数			事务成功率（%）			每秒点击率	平均流量（字节/秒）
	业务1	业务2	业务3	业务1	业务2	业务3	业务1	业务2	业务3	业务1	业务2	业务3		

并发用户数与数据库服务器					
并发用户数	CPU 利用率	内存利用率	磁盘 I/O 参数	DB 参数	其他参数

Offer

续表

并发用户数与应用服务器			
并发用户数	CPU 利用率	内存利用率	磁盘 I/O 系数

3. 疲劳强度与大数据量测试

（A）疲劳强度测试

极限名称				
前提条件				
运行时间				
测试方法				
并发用户数	输入/动作	是否正常运行	故障时间点	故障描述
	核心业务 1			
	核心业务 2			
	典型业务 1			
	典型业务 1			

（B）大数据量测试

功能						
目的						
方法						
并发用户数与事务执行情况						
用例说明	事务平均响应时间（秒）	事务最大响应时间（秒）	平均每秒处理事务数	事务成功率（%）	每秒点击率	平均流量（千字节/秒）

4. 网络性能测试

目的	测试系统运行在不同网络带宽下的性能情况以及与并发用户数量的关系			
方法	不同广域网带宽下以 LoadRunner 录制脚本，以不同带宽和用户数进行压力测试，并记录各事务的响应情况、路由器端口的流量和其他数据			
运行时间	10 小时			
并发用户数	带宽	各事务响应情况	路由器端口流量	丢报率



续表

5. 服务器性能初级测试

不用或写少量用例，使用用户并发测试、疲劳强度与大数据量测试时总结的数据。

		CPU 利用率	内存利用率	磁盘利用率
用户并发测试	数据库服务器			
	Web 应用服务器			
疲劳强度测试	数据库服务器			
	Web 应用服务器			
大数据量测试	数据库服务器			
	Web 应用服务器			

9.6.2 英文模板

Performance Test Case Template

1. Anticipant Objective Test

Case Sequence				
Performance Description				
Case Objective				
Precondition				
Particular Rule Explanation				
Dependence among Cases				
Step	Input/Action	Anticipant Performance（average）	Actual Performance（average）	Regression Test
1.				
2.				
3.				

2. oncurrent User Test

（A）Core module A

Function	
Objective	
Method	
The Relations between Number of Concurrent Users and Transaction Execution Situation	

Offer!

续表

Concurrent User Number	Average Transaction Response Time (seconds)	Maximum Transaction Response Time (seconds)	Transactions per second	Successful Percent of Transactions(%)	Hits per second	Average Throughput (bytes/second)

The relations between number of concurrent users and database server					
Concurrent User Number	% Processor Time	Memory Utilization Ratio	Disk I/O Parameters	DB Parameter	Other Parameters

The Relations between Number of Concurrent Users and Application Server			
Concurrent User Number	% Processor Time	Memory Utilization Ratio	Disk I/O Parameters

（B）Combined Businesses

Function	
Objective	
Method	

The Relations between Number of Concurrent Users and Transaction Execution Situation														
Concurrent User Number	Average Transaction Response Time (seconds)			Maximum Transaction Response Time (seconds)			Transactions per second			Successful Percent of ransactions(%)			Hits per second	Average Throughput(bytes/second)
	Business1	B1	B3	B1	B2	B3	B1	B2	B3	B1	B2	B3		

The Relations between Number of Concurrent Users and Database Server					
Concurrent User Number	% Processor Time	Memory Utilization Ratio	Disk I/O Parameters	Concurrent User Number	% Processor Time

The Relations between Number of Concurrent Users and Application Server

Offer

续表

Concurrent User Number	% Processor Time	Memory Utilization Ratio	Disk I/O Parameters

3. Fatigue Strength and Large Data Volume Test

（A）Fatigue Strength Test

Extremity Test Denomination				
Precondition				
Runtime				
Test Method				
Concurrent User Number	Input/Action	Pass or Fail	Malfunction Time Point	Malfunction Description
	Core Business 1			
	Core Business 2			
	Typical Business 1			
	Typical Business 2			

（B）Large Data Volume Test

Function						
Objective						
Method						
The Relations between Number of Concurrent Users and Transaction Execution Situation						
Case Explanation	Average Transaction Response Time（seconds）	Maximum Transaction Response Time（seconds）	Transactions per second	Successful Percent of Transactions(%)	Hits per second	Average Throughput（KB/s）

4. Network Performance Test:

Objective	Test the system performance situation of different network bandwidth, the relations between the performance and concurrent user number.
Method	Apply LoadRunner to record scripts with different WAN bandwidth, process stress test with different bandwidth and concurrent users, record the response of each transaction and other data such as router port flux.
Runtime	10 hours



续表

Concurrent User Number	Bandwidth	Response of Each Transaction	Router ports Flux	Package Losing Radio

5. Elementary Performance Test of Servers

It is not necessary to use test cases here or use only a small quantity. The test results of concurrent user test, fatigue strength and large data volume test can be used here.

		% Processor Time	Memory Utilization Ratio	Disk Utilization Ratio
Concurrent User Test	Database Server			
	Web Application Server			
Fatigue Strength Test	Database Server			
	Web Application Server			
Large Data Volume Test	Database Server			
	Web Application Server			

9.7 缺陷记录报告模板

缺陷记录报告是测试人员的重要工作成果，有缺陷记录、缺陷分类、缺陷跟踪 3 大功能。开发人员和项目领导是缺陷报告的主要读者。

书写清晰、完整的缺陷报告是对保证缺陷正确处理的最佳手段，同时也保证了工程师及其他质量保证人员后续工作的顺利开展。

缺陷报告通常由下列几部分组成。

- 缺陷的标题。
- 缺陷的基本信息。
- 测试的软件和硬件环境。
- 测试的软件版本。
- 缺陷的类型。
- 缺陷的严重程度。
- 缺陷的处理优先级。



- 复现缺陷的操作步骤。
- 缺陷的实际结果描述。
- 期望的正确结果描述。
- 注释文字和截取的缺陷图像。

书写缺陷记录报告的准则包括。

- 一个报告只描述一个缺陷。
- 保证重视缺陷。
- 注明姓名和日期。
- 及时报告缺陷。
- 分析缺陷，使用必要的最少步骤来重现缺陷。
- 分步描述，方便阅读。
- 语气谦虚谨慎。
- 一定要报告不可重现的错误。
- 不擅自修改他人报告，只能在后面附上自己意见。
- 不夸大缺陷的严重性。
- 不容忍小缺陷。

9.7.1 中文模板

缺陷编号					
软件名称		小版本号		版本号	
测试人员					
日期					
硬件平台				操作系统	
缺陷描述					
详细描述	1. 2. 3. 4. 预期结果： 实际结果：				
备注					

9.7.2 表格项解释

缺陷编号：缺陷跟踪系统如 TD、BUGZILLA 会自动给出编号。

小版本号：一个大版本经过小变化，但尚不足以被称为另一个大版本，此时的版本号称为小版本（Build）号。

Offer

硬件平台：指 CPU 等硬件资源。如 Intel 公司的 Strong Arm 系列及 Motorola 公司的 DragonBall 系列。

缺陷描述：缺陷的一句话的概述。作用是使开发人员看一眼就能明白大致情况。

预期结果：按照需求应当出现的正确结果。

实际结果：实际上出现的错误。与预期结果相对照，清楚地说明了缺陷现象。

备注：可以包含测试人员对错因的分析定位。主要包括复现步骤中可能引起混乱的补充信息，是对操作步骤的进一步描述，例如：

- 截取缺陷特征图像文件，在图 9-1 中可用画笔程序（用红色）圈出问题所在。此时在“输入编号”文本框中输入 f，软件系统却将编号判断为 0。用红笔圈出后，一目了然。这样，可以更好地帮助程序员定位错误。

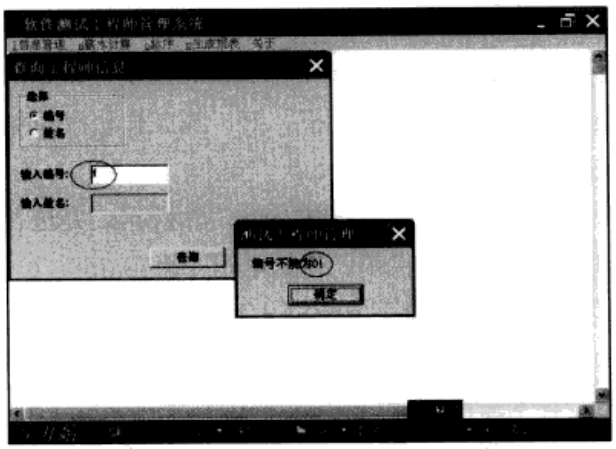


图 9-1 缺陷特征图像文件

- 测试过程需要使用的测试文件。
- 测试附加的打印机驱动程序。
- 再次描述重点，避免开发人员将缺陷退回给测试人员补充更多信息。
- 再次指明该缺陷是否在前一版本已经存在。
- 多个平台之间是否具有不同表现。
- 注释包含缺陷的隔离信息，指出缺陷的具体影响范围。

9.7.3 英文模板

Defect number					
Dynacomm		Build number		Version number	
Testers					
Date					
Hardware platform				Operating system	
Defect summary					

Offer

续表

Defect details	1. 2. 3. 4. Anticipative result: Actual result:
Annotation	

9.8 例会记录模板

例会的作用是保持良好的、有效的沟通，可以及时发现测试中的问题或不正常的现象。整个项目组的例会通常是安排在每周周一或周五，对一周工作进行总结；针对度量结果，对接下来的工作进行规划，进行本周详细工作安排。

项目负责人要在每周例会上收集项目组成员的工作数据，并进行汇总分析，同时应定期召开部门例会，解决工作中的问题，提供部门内的学习平台。

通常由项目负责人主持例会，由自己或指定某下属记录会议情况。注意，会议记录必须存档。

每周例会上的工作安排一般只限于本周或是过后的两三天，时间不会太长，对长时间工作的安排容易失去精确性，且不易控制。

对于测试小组，甚至每日都有例会，由组长布置目前的工作。例会由组长主持，通常由组长自己做会议记录，有时也会指定一个组员负责记录。注意，会议记录也必须存档。

9.8.1 中文模板

例会记录

测试组编号		例会时间	
主持人			
会议记录人			
参加人员			
例会议题	1. 2. 3.		
议题讨论结果			
1. 2. 3.			

Offer!

9.8.2 英文模板

Regular Meeting Record

Test Group Number		Meeting Time	
Emcee			
Recorder			
Attendee			
Discussion Topics	1. 2. 3.		
Discussion Conclusions			
1. 2. 3.			

9.9 工作日志模板

工作日志记录软件测试工程师每天的工作，是企业实施过程化管理的重要内容，是领导掌握员工情况和做决策的重要依据。

测试日志用来记录每天测试执行的情况：其中主要需要记录用例执行情况，这是为了进行测试进度度量，帮助测试经理了解各测试工程师的进展是否一致，合理调整测试资源，及时对测试中的进度风险进行控制；同时，测试日志还需要记录被测对象当天的缺陷发现情况，这可以帮助测试经理了解被测对象的缺陷发展走势，通过缺陷分析判断是否可以退出测试。

工作日志的内容主要包括。

（1）当天测试执行情况的概述。

（2）用例执行统计（包括总用例数、计划当天执行用例数、当天实际执行用例数、累计已执行用例数、用例通过情况以及各用例执行时间等），这实际上是对于进度和用例执行效率的度量。

（3）缺陷发现情况统计（包括当天发现的各级别缺陷数、累计已发现的各级别缺陷数以及各级别缺陷统计等），这实际上是对缺陷情况的度量，可以用来了解缺陷发展的趋势。

（4）当天测试过程中遇到的困难（便于及时反应情况，做好沟通交流，解决问题）。

日志通常在每个工作日下班前被提交。

9.9.1 中文模板

工作日志

姓名		测试组编号		日期	
当日工作任务					
1. 2. 3.					

Offer

续表

完成情况（%）	
工作成果	
1. 2. 3.	
遇到的问题	
1. 2.	

9.9.2 表格项解释

当日工作任务：当天的任务的量化描述。

完成情况：当天任务的总的完成情况的百分比。

工作成果：包括用例执行统计和缺陷发现情况统计，各任务完成的百分比。

遇到的问题：当天的一些工作上的疑难。

9.9.3 英文模板

Daily Work Record

Name		Test Group Number		Date	
Task Description					
1. 2. 3.					
Accomplished（%）					
Achievements					
1. 2. 3.					
Problems and Questions					
1. 2.					

9.10 周报模板

周报是一种非常有效的工具，它可以使领导对员工的具体工作有所把握，及时发现员工的思想倾向和问题，也能多角度反映出项目的进度，同时也使员工养成定期自我反省和总结的习惯。

周报书写的要点如下。

Offer

- 使用标准的周报模板，按规范要求填写。
- 领导可以适当进行点评，从内容到形式指出其中的不足，同时不要忘了对写得精彩的地方要给以充分的肯定。
- 不要为了写周报而周报，应将周报当做是一种工具和过程。
- 通常来说，周报在每周五下午被提交。

9.10.1 中文模板

2010 年 9 月 10 日周报

时　间	工作事项简述	持续时间	收效和成果
本周：09/06 ~09/10	1.		
	2.		
	3.		
	4.		
下周：09/13 ~09/17	1.		
	2.		
	3.		
	4.		
问题和意见			

9.10.2 表格项解释

- 工作事项简述：每项工作任务的概述。
- 持续时间：进行该事项的时间，如 3 天，即从周二上午到周四下午。
- 收效和成果：取得的进步，最好量化一下，如 100%完成。
- 问题和意见：对工作中所遇问题的描述，提请领导注意。
- 下周：之所以包含下周工作的描述，是对未来工作的计划和预测，使领导心中有数并能够及早发现问题。

9.10.3 英文模板

Weekly Report on September 10, 2010

Time	Work Action Description	Duration	Progress and Result
This week: 09/06 ~09/10	1.		
	2.		
	3.		
	4.		
Next week: 09/13 ~09/17	1.		
	2.		
	3.		
	4.		
Problems or Thoughts			



9.11 单元测试报告模板

图 9-2 所示的数据摘自《实用软件度量》(Capers Jones，McGraw-Hill 1991)，它列出了准备测试、执行测试和修改缺陷所花费的时间(以一个功能点为基准)，这些数据显示单元测试的成本效率大约是集成测试的两倍、系统测试的 3 倍。

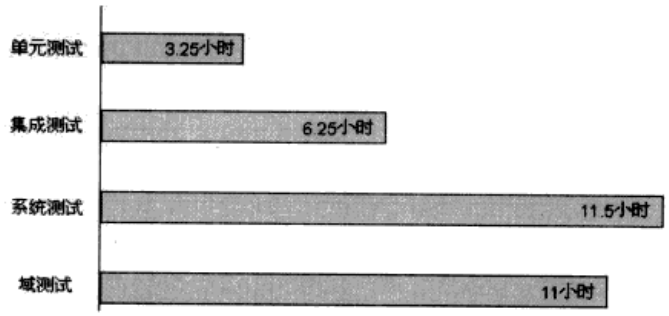


图 9-2 单元测试与集成、系统、域测试的成本效率对比

从图 9-2 中我们可以看出，尽可能早地排除尽可能多的 Bug 可以大大减少后阶段测试的费用。

图 9-2 中的域测试(Field Test)，意思是在软件投入使用以后，针对某个领域所作的所有测试活动。

单元测试报告是总结单元测试成果的总结性报告,在报告中主要给出编写单元测试报告的目的，被测单元的描述、单元结构、单元测试的过程、日期、设计的测试用例、通过/失败的测试统计、对失败的用例进行分析、单元测试存在的不足以及还需要完善的地方等。

单元测试报告是单元测试的主要产品，主要附件还包括缺陷报告。

单元测试的原则如下。

- 过于简单的方法不必测试。
- 测试任何可能出错的地方。
- 使测试用例可重用。
- 重视测试类的接口甚于类的实现。
- 不能依赖固定的类方法调用顺序。

通常来说，大部分单元测试的工作仍是开发人员在做，而不是测试工程师。

9.11.1 中文模板及各项解释

1. 编写目的

- 对单元测试结果进行整理和汇总，以形成正式的测试文档。
- 为软件产品的评审验收提供依据。

Offer!

- 纳入软件产品配置管理库。

2. 软件单元描述

简单描述被测试单元或相关单元的产品项目名称、所属子系统、单元要完成的功能、需求和设计要求等。

3. 单元结构

画出本单元的组织结构，包括本单元包括的属性、方法、输入/输出等。

4. 单元控制/时序流图

画出本单元的控制结构或操作时序，并描绘其大概过程。

5. 测试过程

简要描述本单元的测试过程。

6. 测试结果

代码审查结果

Bug ID	审查人员	审查日期	问题描述

测试用例统计：

测 试 项	测试用例号	测试特性	用例描述	测试结论	对应 Bug ID

单元测试产品提交

- 提交测试覆盖率编译运行时的.gcov 文件。
- 提交存储检查结果的.ccmalloc 文件。
- 提交性能分析时编译运行的.gprof 文件。
- 利用-O0、-O2、-O3 等 3 种编译优化选项编译被测代码时产生正确性测试结果的文件。
- 提交驱动模块、桩模块和测试用例对应的源代码、注释，注意，要与测试用例中的测试用例号相对应。
- 提交加载测试用例编译运行后的.h 和.cpp 或.c 文件及 makefile 文件。
- 在单元测试中提交的软件 Bug 清单。
- 本单元测试报告。



7．质量评估

对本测试单元模块的评价，包括功能、性能、余量、人机交互界面、可靠性及可维护性等。

8．总结

对本次测试进行简单的总结陈述。

9.11.2 英文模板

Unit Test Report Template

1．Objective

2．Unit Description

3．Unit Configuration

4．Unit Control/Time Sequence Flow Chart

5．Testing Process

6．Testing Results

- Code Checkup Results：

Bug ID	Censor	Checkup Date	Defect Description

Testing Case Statistics：

Testing Item	Testing Case Number	Testing Properties	Case Description	Testing Conclusion	Corresponding Bug ID

Unit Testing Deliveries

7．Quality Evaluation

8．Summary

9.12 测试总结模板

测试总结用来总结测试活动的结果，并根据这些结果进行评价。在测试的各个阶段完成后，我们都可以提交测试总结报告，总结本阶段产品的缺陷，提出改进和建议，决定是否可以进入

Offer

下一阶段。测试总结分析测试结果，描述系统是否符合需求。预期参考人员包括用户、测试人员、开发人员、项目管理者、其他质量管理人员和需要阅读本报告的高层经理。

通常，用户对测试结论部分比较感兴趣，开发人员希望从缺陷结果及分析得到产品开发质量的信息，项目管理者对测试执行中成本、资源和时间给予重视，而高层经理希望能够阅读到简单的图表并且能够与其他项目进行同向比较。此部分可以具体描述为什么类型的人可参考本报告 XXX 页 XXX 章节，报告读者越多，工作就越容易被重视。

测试结果及缺陷分析是整个测试报告中最激动人心的部分，这部分主要汇总各种数据并进行度量，度量包括对测试过程的度量和能力评估以及对软件产品的质量度量和产品评估。对于不需要过程度量或相对较小的项目，例如用于验收时提交用户的测试报告、小型项目的测试报告，可省略过程方面的度量部分；而采用了 CMM/ISO 或其他工程标准过程的，则需要提供过程改进建议和参考的测试报告，并列出过程度量，主要用于公司内部测试改进和缺陷预防机制。

在测试结果及缺陷分析中，我们可按照问题的严重性、类型和状态对缺陷进行分类。根据对缺陷的分类，可以利用 Excel 生成图形，更直观地表现出缺陷分布。以某软件测试工程师管理系统项目为例，基于功能模块分类的缺陷分布图如图 9-3 所示。

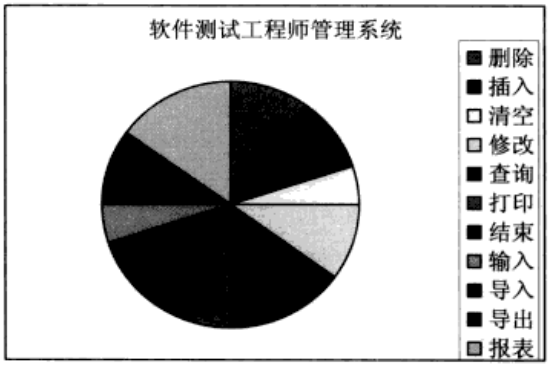


图 9-3 基于功能模块分类的缺陷分布图

测试结论包括。

（1）测试执行是否充分（可以增加对安全性、可靠性、可维护性和功能性描述）。

（2）对测试风险的控制措施和成效。

（3）测试目标是否完成。

（4）测试是否通过。

（5）是否可以进入下一阶段项目目标。

建议包括。

（1）对系统存在问题的说明，描述测试所揭露的软件缺陷和不足，以及可能给软件实施和运行带来的影响。

（2）可能存在的潜在缺陷和后续工作。

（3）对缺陷修改和产品设计的建议。

Offer!

（4）对过程改进方面的建议。

有些测试报告也可能而按照主题逐项列出测试项、缺陷、分析和建议，尤其是在第三方评测报告中。

9.12.1 中文模板及表格项解释

测试总结

<table>
<tr><td colspan="2">测试期产品名称</td><td colspan="5"></td></tr>
<tr><td colspan="2">测试类型</td><td colspan="5">□系统测试 □用户测试 □评估产品测试 □母盘内容更换测试</td></tr>
<tr><td colspan="2">产品正式名称</td><td colspan="5">中文：
英文：</td></tr>
<tr><td colspan="2">产品简称(中文、英文)</td><td colspan="2"></td><td>开发组</td><td colspan="2"></td></tr>
<tr><td>版本号：</td><td></td><td>构造号</td><td></td><td>语种</td><td colspan="2">中文/英文</td></tr>
<tr><td colspan="2">项目组开发工作量：(人日)</td><td colspan="5">人日</td></tr>
<tr><td colspan="2">本次版本升级修改过的代码行数（千行）（含增加、删除、修改）：</td><td colspan="5">千行</td></tr>
<tr><td colspan="2">产品总的代码行数（千行）：</td><td colspan="5">千行</td></tr>
<tr><td colspan="7">产品简介：</td></tr>
<tr><td colspan="2">产品建议配置：</td><td colspan="5"></td></tr>
<tr><td colspan="2">产品组成模块清单：</td><td colspan="5"></td></tr>
<tr><td colspan="2">产品(主要)组成文件清单：</td><td colspan="5"></td></tr>
<tr><td colspan="2">产品组成介质清单：</td><td colspan="5">中文版：
英文版：</td></tr>
<tr><td colspan="2">实际测试时间段：</td><td colspan="5"></td></tr>
<tr><td colspan="2">实际测试人员：</td><td colspan="5"></td></tr>
<tr><td colspan="2">实际测试工作量：(人日)</td><td colspan="5">人日</td></tr>
<tr><td colspan="2">实际测试环境：</td><td colspan="5"></td></tr>
<tr><td colspan="7">测试活动简述：</td></tr>
<tr><td rowspan="3">测试错误分类</td><td></td><td>死机</td><td>致命</td><td>严重</td><td colspan="2">一般</td></tr>
<tr><td>发现</td><td></td><td></td><td></td><td colspan="2"></td></tr>
<tr><td>改正</td><td></td><td></td><td></td><td colspan="2"></td></tr>
<tr><td rowspan="2">软件继承性</td><td colspan="2">继承软件名称：</td><td></td><td>版本号：</td><td colspan="2"></td></tr>
<tr><td colspan="6">对继承软件的改进/改变处：
Bug 的修改：
功能的改进：
新功能的增加：
为配合相关的插件或软件做的改动：
针对某些功能模块做的优化：
其他：</td></tr>
</table>

续表

软件遗留问题（发生原因、对功能有何影响、解决措施）：		
测试组测试结论：		
产品度量	测试期发现的 Bug 共　　个；建议共　　个,采纳　　个。 Bug 密度（系统测试的 Bug 数/修改的代码行数）：　　个/千行 Bug 密度（系统测试的 Bug 数/总的代码行数）：　　个/千行	
	待发行产品遗留 Bug 共　　个；严重及严重以上遗留 Bug 共　　个。 Bug 密度(遗留 Bug 数/修改的代码行数)：　　个/万行 Bug 密度（遗留 Bug 数/总的代码行数）：　　个/万行 Bug 密度（严重及严重以上遗留 Bug 数/总的代码行数）：　　个/万行	
测试综合报告评审	评审方式、人员、时间：	
	评审结论：	
测试经理	签名：	日期:　　年　　月　　日
系统分析员	意见：	
	签名：	日期:　　年　　月　　日
开发经理	意见：	
	签名：	日期:　　年　　月　　日
项目经理	意见：	
	签名：	日期:　　年　　月　　日
备注：		

9.12.2 英文模板

Test Summary

Informal Product Name(test period)					
Testing Type	□System Test □User Test □Product Evaluation Test □Mother-Disk Content Replacement Test				
Official Product Name	Chinese: English:				
Product Name Abbreviation (Chinese/English)				Development Team	
Version No.		Build No.		Language	Chinese/English
Product Team Development Workload：(person days)					
Code Line Number of Updating and Modification at This Time（KLOC）(including Increasing, Deletion, Modification)					
Total Code Line Number(KLOC)					
Product Brief:					

续表

Proposed Configuration					
Product Module list					
Main File List					
Product Medium List	Chinese Version: English Version:				
Actual Test Period					
Testers					
Test Workload (person days)					
Test Environment					
Testing Activity Narration:					
Defect Sorting		Dead	Fatal	Serious	Ordinary
	Found				
	Modified				
Software Inheritance	Inherited Software Name			Version No.	
	Improvement/change comparing to the inherited software: Bug improvement: Function improvement: Adding new functions: Changes for matching relative plug-in or software: Optimization for some function modules: Others:				
Reserved Software Defects (reason, influence, solution):					
Test Team Testing Conclusions:					
Product Measurement	Totally __ bugs; Totally __ suggestions, we adopted __ suggestions. Bug Density (bug number in system test/modified code line number): __ bugs/KLOC Bug Density (bug number in system test/total code line number): __ bugs/KLOC				
	Reserved software bugs:___ ; Serious or more: ____ bugs. Bug Density(reserved bug number/ modified code line number): ___ bugs/10KLOC Bug Density(reserved bug number/ total code line number): ____bugs/10KLOC Bug Density(reserved serious or more serious bug number/ total code line number): _____bugs/10KLOC				
Synthesis Test Report Review	Review manner, attendee, time:				
	Review Conclusions:				
Test Manager	Sign:	Date: Month Day Year			
System Analyst	Opinion:				
	Sign:	Date: Month Day Year			
Development Manager	Opinion:				
	Sign:	Date: Month Day Year			
Project Manager	Opinion:				
	Sign:	Date: Month Day Year			
Note:					

